iTAP Service Processor
Chip Specification

Appendix A
10/00

AUTHOR:_____

REVISION: 0.3

DATE: 31.AUG.2000

APPROVED:_____

1 Overview

The iTAP Port Processor chip is a communications processor which extracts and maps TDM, ATM and IP payload data from and to SONET interface signals and a UTOPIA 2/3 ATM/POS interface.

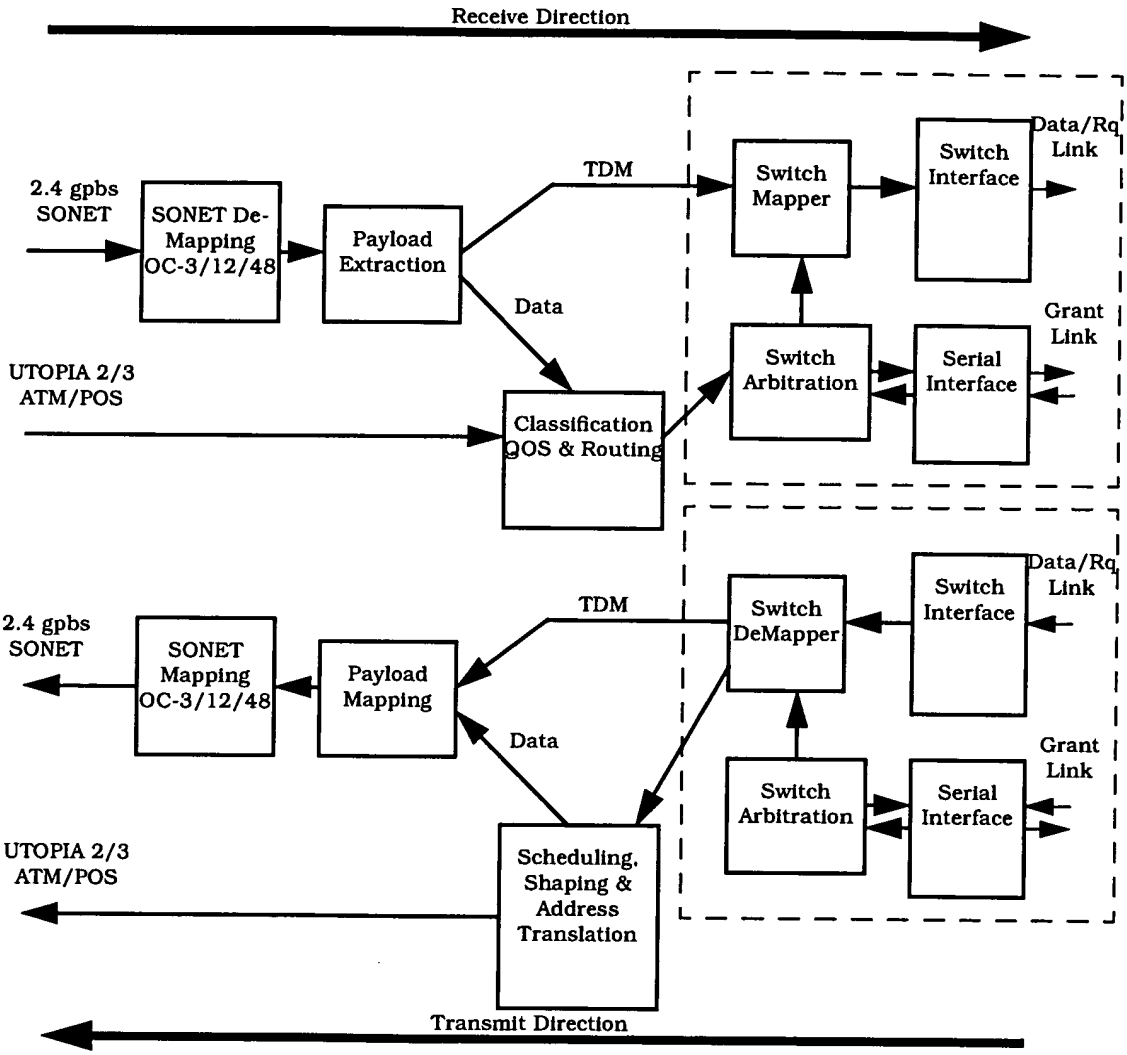


Figure 1: iTAP Port Processor Overview

As shown in Figure 1, data is received on two types of interfaces: SONET and UTOPIA. There are 4 separate SONET interfaces. These interfaces can be configured as 4 OC-3s or 4 OC-12s or a single OC-48. Transport and Path overhead temination functions are supported. There is an external serial DCC port provided to pass the DCC overhead bytes to an external unit.

The total of SONET interfaces is four and can not exceed 2.4 gbps. The single UTOPIA Interface can be configured as Level 2 or Level 3, ATM or packet. The UTOPIA interface can be used in addition to the SONET interfaces, but the aggregate bandwidth of all interfaces into the Port Processor can not exceed 2.4 gbps unless the switch interface is **not** used as would be the case in the single chip router application (this is discussed later in this section).

Payload data is extracted from the SPEs of the incoming SONET signals. The type of payload in each SPE is provisioned through the external microprocessor interface. TDM data is routed directly to a Switch Mapper and the ATM and IP data is extracted using cell delineation and HDLC processing and then routed to an internal microprocessor for further service. The UTOPIA POS-PHY data, ATM cells or variable-length packets, is routed directly to a series of internal traffic processors. These traffic processors perform connection ID validation, high-speed IP Forwarding, ATM VPI/VCI lookup, traffic classification, traffic policing and congestion control.

All of the receive traffic is destined for the Switch fabric and is mapped into an OneX proprietary row format. This row consists of NNN slots of 36 bits each. Each slot carries 4 bytes of data and 4 bits of control information. TDM data is allocated dedicated bandwidth through the switch fabric and the OneX proprietary row format is designed to optimally support TDM traffic down to the VC-11 and VC-12 level. Incoming TDM traffic is never buffered, it is routed directly to pre-allocated and pre-configured slots in the outgoing rows. ATM cells and PPP frames are not allocated dedicated bandwidth through the switch fabric. The bandwidth for these ATM and IP data units must be arbitrated through the switch and the destination Port Processor. The row is designed to support a super-slot or data-slot which is an aggregation of 16 single slots.

The switch row is serialized and distributed across high speed serial ports for transmission to the switch. The aggregate throughput to the switch is N.NN gbps.

In the Transmit Direction, the Port Processor responds to arbitration requests from a Switch chip. The arbitration grant from the Port Processor is based on the available buffer space in the traffic memory. TDM traffic and granted data traffic is received through the high speed switch interface. TDM traffic is mapped directly into outgoing SPEs. ATM Cells and PPP frames are all buffered externally where they wait to be scheduled out a transmit SONET or UTOPIA interface. There is an internal microprocessor that is dedicated to processing the transmit data units. This processor is responsible for scheduling, shaping and address translation. Once the data is scheduled, it is mapped into a SONET SPE or is routed to the UTOPIA interface.

OC-3, OC-12 and OC-48 frames are generated and Transport and Path overhead generation functions are supported in the Port Processor. An external serial port is available for DCC input. As in the receive direction, the total of SONET interfaces is four and can not exceed 2.4 gbps. The single UTOPIA Interface can be configured as Level 2 or Level 3, ATM or packet. The UTOPIA interface can be used in addition to the combination of SONET and Telecom Bus interfaces, but the total aggregate bandwidth out of the Port Processor can not exceed 2.4 gbps except when the switch interface is **not** used.

There is a data path connection between the SONET interfaces and the UTOPIA Interface. This is provided to enable a single chip routing function. In this mode no Switch chips are required.

The Port Processor can store a total of 50 ms worth of data received at an OC-48 rate. This equates to 2.488 gbps/50ms = 124,400 Mbits = ~16MBytes. This 16 MBytes of memory is split between the Rx side and the Tx side of the PP. The ratio of the split is TBD.

The Port Processor is able to process packets and cells at an OC-48 rate. This equates to (2.488 gbps X 86/90)/(48bytes/packet X 8bits/byte) = 6.19 Mpackets/s = 160ns and for cells (2.488 gbps X 86/90)/(53bytes/packet X 8bits/byte) = 5.61 Mcells/s = 178ns.

The host processor interface is based on mailboxes and is described in detail in a later section .

1.1 Features

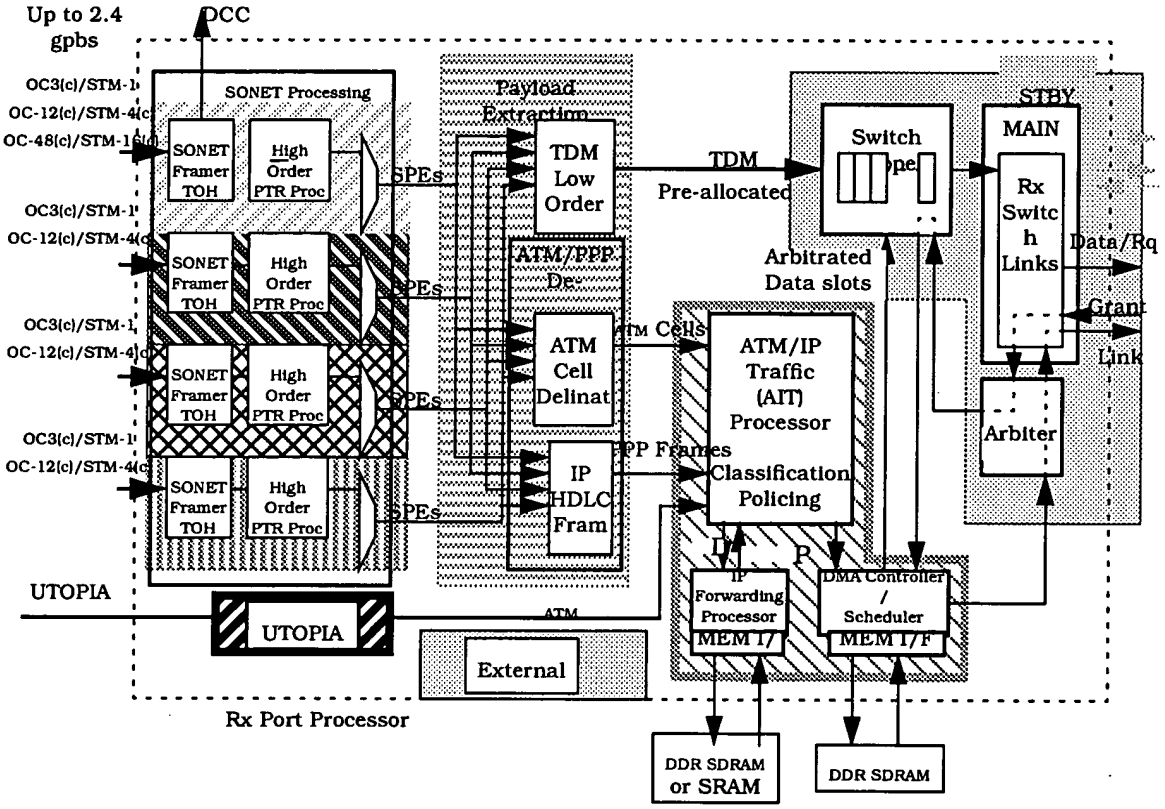
- **Line Interfaces**

1. SONET
   - 4 X OC-3
   - 4 X OC-12
   - 1 X OC-48
2. Telecom Bus
   - Parallel(?)
   - Serial
3. UTOPIA
   - Level 2/3
   - ATM / Packet
   - 100 MHz

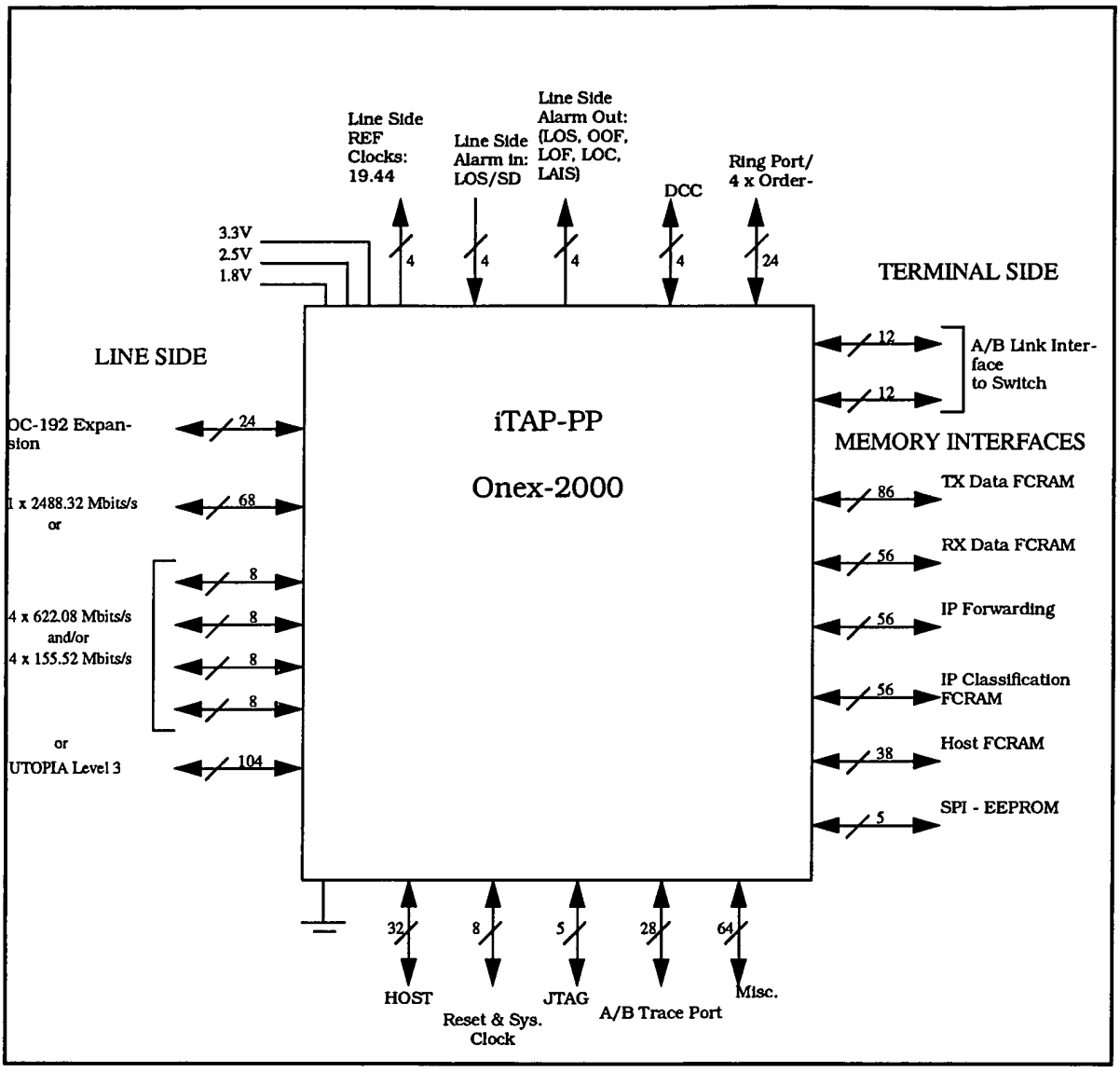
See the feature list in the Product Outline...

2 Synchronization

The timing domains are shown in the following figures. Each signal that crosses a timing domain boundary needs to be handled carefully to assure that no ill effects of meta-stability will be seen. Single bit signals are double sampled in the destination timing domain. Multi-bit buses will be sampled in the destination timing domain after a single bit asynchronous enable signal indicates that the data bus is stable.



3 External Port Descriptions



3.1 Utopia L3 Port

32 bit wide Utopia L3 interface configurable for Master or slave.

3.2 OC-3/12/48

On the line side the PP supports four interfaces. The PP can be run in a single chip application. For a single PP application the total SDH/SONET/Telecom Bus interface can be 2.5Gbits/s interfacing to the Utopia port.

1. A single sixteen bit 155MHz LVPECL parallel interface to support one STS-48/48c or STM-16/16c.
2. Four 622/155Mhz serial LVPECL interfaces to support either STS-12/12c, STM-4/4c, STS-3/3c and/or STM-1.
3. One Utopia L3 interface to support ATM and IP traffic.

3.3 OC-192/STM-16 Expansion Port

The OC-192/STM-16 expansion port will allow for connection of four iTap PP. An external multiplexer will be required to mux the four OC-48 signals to OC-192.

3.4 Line Side ReferenceClocks

In the transmit direction the PP can be configured to provide self-time and loop-time on a per-port basis. Also, the PP has four reference clock outputs, each corresponding to a SDH/SONET interface. A configuration register selects the frequency of these clocks. The clock rates are a divided down 8Khz or 19.44Mhz from the received clock.

3.5 Alarm In

RESERVED

3.6 Alarm Out

RESERVED

3.7 DCC

RESERVED

3.8 Ring Port and Orderwire (Still Defining)

RESERVED

3.9 Terminal Side Switch Links

RESERVED

3.10 TX and RX Data FCRAM port Interfaces

RESERVED

3.11 IP Forwarding and Classification FCRAM

RESERVED

3.12 FCRAM Host Interface Memory

RESERVED

3.13 Serial Boot Prom

RESERVED

3.14 Miscellaneous Pins

RESERVED

3.15 Internal Processor Trace Port

RESERVED

3.16 JTAG

RESERVED

3.17 Rest & System Clock

RESERVED

3.18 Host Interface

RESERVED

3.19 Power Pins

The PP uses 1.8V for core operation with 1.8/2.5/3.3V for I/Os.

4 Data Path Description

RESERVED

4.1 Receive Data Path

RESERVED

4.1.1 Receive Side Block Diagram

4.1.2 Receive Side SONET Interfaces

- RESERVED

4.1.2.1 Serial to Parallel Conversion

RESERVED

4.1.2.2 SONET Framing

RESERVED.

4.1.2.3 Transport Overhead Termination

RESERVED

4.1.2.4 Pointer Processing (H1, H2, H3)

RESERVED

4.1.3 Receive Side SONET Interfaces

.RESERVED

4.1.3.1 Serial to Parallel Conversion

RESERVED

4.1.3.2 SONET Framing & Descrambling

RESERVED

4.1.3.3 Transport Overhead Termination

RESERVED

4.1.3.4 OC-48 Data and Timing Multiplexors

RESERVED

4.1.3.5 High Order Pointer Tracking (H1, H2, H3)

RESERVED

4.1.4 Receive Side SONET Overhead Processing

RESERVED

4.1.4.1 Transport Overhead Termination

RESERVED

4.1.4.1.1 Orderwire

RESERVED

4.1.4.1.2 Section & Line DCC

RESERVED.

4.1.4.2 Path Overhead Processor

RESERVED

4.1.5 OC-48c

RESERVED

4.1.6 Receive Side Telecom Bus I/F

RESERVED

4.1.7 SPE Mux

RESERVED.

4.1.8 SPE Processing

RESERVED

4.1.8.1 Payload Extraction

RESERVED

4.1.8.1.1 TDM Demultiplexing from SONET/SDH

RESERVED

4.1.8.1.2 ATM Extraction From SONET/SDH

RESERVED

4.1.8.1.3 IP Extraction Out of SONET

RESERVED

5 Descriptor Builder

5.1 Overview

RESERVED

5.2 Functions

RESERVED

5.3 Input Data Structure

RESERVED

5.4 Output Data Structure

RESERVED

5.5 Diags, Test Modes and "T" bytes

RESERVED

5.6 Input Traffic Types

RESERVED

5.6.1 ATM & AAL5 Cells

RESERVED

5.6.2

RESERVED

5.6.3 IPv4

RESERVED

5.6.4 IPv6

RESERVED

5.6.5

RESERVED

5.6.6 MPLS

RESERVED

5.6.7 Frame Relay

RESERVED

5.7 TIE Instructions:

RESERVED.

6 Traffic Forwarding and Classification

Traffic forwarding and classification block (TFC) performs the function of IP packet layer 3 route lookup, IP classification, ATM cell VPI/VCI lookup, Frame Relay DLCI lookup and MPLS label lookup. The design goals and worst case processing times are listed in Table 6-1, "Design Goals," on page 27.

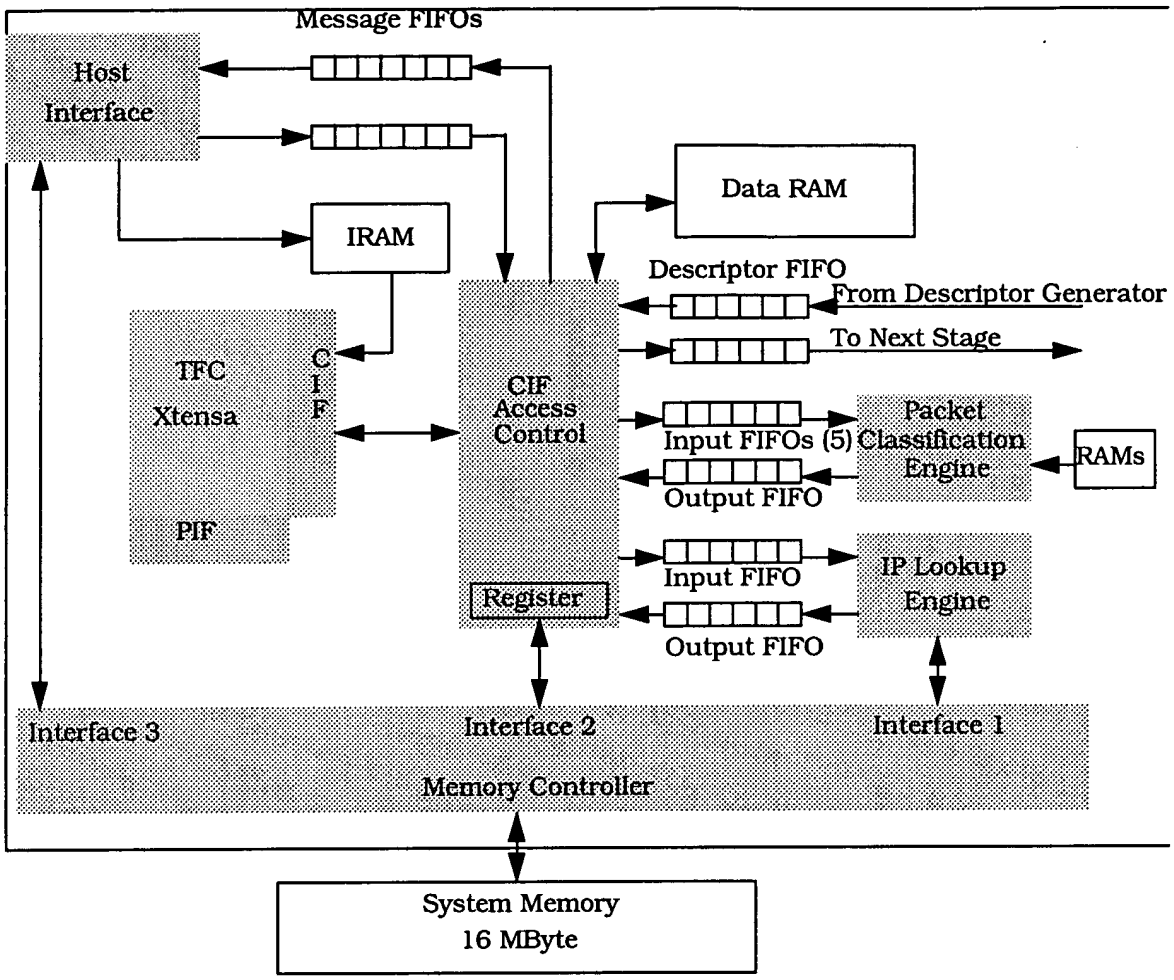
Table 6-1: Design Goals

Traffic Type	# of Flows / Routing Entries	Line Rate	Cell/Pkt Rate @ MIN PDU	Processing time
ATM cell	upto 128K	2.488 Gbps	5.61M cps	180 ns
IP packet	upto 128K	2.488 Gbps	6.19M pps	160 ns
FR packet	upto 128K	2.488 Gbps	6.19M pps	160 ns
MPLS packet	upto 128K	2.488 Gbps	6.19M pps	160 ns

Note that for each traffic type, upto 128K flows are supported, however, the sum of flows from all four traffic types shall not exceed 128K.

6.1 Data Flow

The following block diagram shows the components of and the data flow through the TFC block.



IP packets, ATM cells, Frame Relay and MPLS packets are merged into one flow at the SONET and UTOPIA interface. TFC block receives a traffic descriptor from the Descriptor Builder for each data chunk PDU (52 bytes). Based on traffic type, the TFC processor performs the corresponding lookup/classification functions. The results of lookup and classification are mergered and combined with fields from incoming descriptor to form a new outgoing descriptor, which is passed to the next stage of packet processing.

There is a mode register, Bypass, which globally turns off the Traffic Forwarding and Classification. In this mode, a external device is used for forwarding and/or classification, and the data descriptors generated are routed directly to the next packet processing stage.

The shaded block are implemented in hardware, please refer to Section 6.10 on page 31 for implementation details.

6.1.1 Inputs to IPF block

RESERVED

6.1.2 Outputs of IPF block

RESERVED

6.2 Overview of IP Routing Table Lookup Algorithms

RESERVED

6.2.1 Multi-bit Trie with controlled prefix expansion

RESERVED

6.2.2 Binary Search on prefix length with Hash tables

RESERVED

6.2.3 Binary(multi-way) Search on prefix ranges

RESERVED

6.3 Search Structure for Multi-bit Trie

RESERVED

6.3.1 Storage requirement and data structure

RESERVED

6.3.1.1 Search Data Structure

RESERVED

6.3.1.2 Determine expansion levels

RESERVED

6.3.1.3 Storage requirement

RESERVED

6.3.1.4 Techniques to reduce storage requirement

6.3.1.4.1 Packed Node

RESERVED

6.3.1.4.2 Leaf Pushing

RESERVED

6.3.2 IP Search database construction

RESERVED

6.3.3 Pseudo-code for Search, Insertion and Deletion

RESERVED

6.3.4 Improvement to Insertion/Deletion scheme

RESERVED

6.3.4.1 Incremental Deletion with reduced binary radix tree

RESERVED

6.3.4.2 Storage analysis for Patricia tree

6.3.4.3 Special case for Class B address update when first stride is 16

RESERVED

6.3.5 TIE Instructions

RESERVED

6.3.5.1 Configuration (State) Registers

RESERVED

6.3.5.2 TIE Instruction Syntax

RESERVED

6.4 IP Packet Classification

The function of packet classification is to indentify the flow a packet belongs to, based on one or more fields in the packet header. The most common fields are the IP destination address(32 bits), IP source address(32 bits), protocol type (8 bits), and TCP/ UDP port numbers (16 bits each) of destination and source applications, and TCP flags. The classification rule set, aka filter database, consists of N rules R_1, R_2, ... , R_N over K fields (dimensions). Each rule R is a K-tuple (F[*1*], F[2], ... , F[*k*]), where F[*i*] is a range (interval) of values the fields *i* may take; each rule also associats with a priority. A query is done for every packet upon the arrival of the packet. A packet $P = [f_1, f_2, ..., f_k]$, where each f_i is a singleton value, matches rule R if for all packet fields *i*, f_i of packet *P* lies in the range F[*i*] of rule R. The packet classification problem is to find the highest priority rule matching a given packet *P*.

6.4.1 Features of IP Classification Function

RESERVED

6.4.2 Theory of Operation

RESERVED

6.4.2.1 Basic Algorithm

RESERVED

6.4.2.2 Algorithm Improvement

RESERVED

6.4.3 Implementation

RESERVED

6.4.3.1 Preprocessing (software)

RESERVED

6.4.3.2 On-line Classification (hardware)

RESERVED

6.4.4 Storage requirement and data structure

RESERVED

6.4.4.1 Search Data Structure

RESERVED

RESERVED

6.11 Hardware Implementation

This section documents the implementation details of all the hardware blocks.

6.11.1 IP Forwarding (IPF)

RESERVED

6.11.2 IP Classification (IPC)

RESERVED

6.11.3 CIF DataRAM Access Control

RESERVED

6.11.3.1 Data RAM (16K Byte)

RESERVED

6.11.3.2 DESC_IN_FIFO

RESERVED

6.11.3.3 DESC_OUT_HH_FIFO and DESC_OUT_LH_FIFO

RESERVED

6.11.3.4 IPF_IN_FIFO

RESERVED

6.11.3.5 BASE_FIFO

RESERVED

6.11.3.6 IPF_OUT_FIFO

RESERVED

6.11.3.7 IPC_IN_FIFOs

RESERVED

6.11.3.8 IPC_OUT_FIFO

RESERVED

6.11.3.9 DESC_TEMP_FIFO_WR and DESC_TEMP_FIFO_RD

RESERVED

6.11.3.10 LOAD_32_CMD

RESERVED

6.11.3.11 LOAD_64_CMD

RESERVED

6.11.3.12 LOAD_RESULT

RESERVED

6.11.3.13 STORE_CMD

RESERVED

6.11.3.14 STORE_DATA

RESERVED

6.11.3.15 FIFO_STATUS

RESERVED

6.11.3.16 RSLT_READY_STATUS

RESERVED

6.11.4 Master(API) Processor Interface

RESERVED

6.11.5 External Memory Interface

RESERVED

6.11.5.1 Format of Commands

RESERVED

7 Receive AIT Processor

The Receive AIT/IP Traffic Processor (Rx AIT Processor) is responsible for managing incoming ATM and IP traffic from the Utopia Interface on the Titan Chip. The IP Forwarding Processor provides various traffic parameter to the Rx AIT Processor through the "Traffic Descriptor". The Rx AIT Processor uses the incoming Traffic Descriptor to decide whether drop/pass/mark incoming cells. This information is placed in an modified output Traffic Descriptor which is passed on to the Data Link Manager for it to decide whether to enqueue the payload that will be forward to the Chiron Switch Fabric.

The Rx AIT Processor is a single Tensilica Processor with 64 bit data bus interface for high data bandwidth and several custom instructions for accelerating the traffic management algorithms. The processor uses an instruction RAM rather than an instruction cache to eliminate instructions stalls from instruction cache misses. The processor uses an on chip data RAM to hold parameters for the traffic management algorithms, and a data cache which is connected to a large external SDRAM memory bank to hold 128K connection tables used for ATM traffic policing. The processor receives a Traffic Descriptor from the IPF Processor from an input FIFO and then generates and updated Traffic Descriptor for the DLM Processor.

The Rx AIT Processor does not directly observe or modify ATM or IP cells, headers or payloads.

7.1 Traffic Descriptor

RESERVED

7.2 Compute Budget

RESERVED

7.3 Processor Memory Subsystem

RESERVED

7.4 Rx_AIT / Rx_DLM Shared Memory

RESERVED

7.5 Rx IP Traffic Management with (W)RED

The Rx AIT Processor use the (Weighted) Random Early Detection, (W)RED, algorithms to manage IP traffic. These algorithms tell the DLM to pass/drop an IP packet before its payload is enqueued by the DLM. The (W)RED algorithms perform both the traffic management function and service differentiation function. This implementation of RED supports 32 Quality of Service buffers, and this implementation of WRED supports 6 precendence levels.

7.5.1 Random Early Detection / RED

RESERVED

7.5.1.1 Drop Probability

RESERVED

7.5.1.2 Average Queue Length

RESERVED

7.5.2 RED Parameters

RESERVED

7.5.3 RED Internal Variables

RESERVED

7.5.4 Random Drop Calculation

RESERVED

7.5.4.1 Weighted Random Early Detection / WRED

RESERVED.

7.5.5 WRED Parameters

RESERVED

7.5.6 Rx_AIT / DLM Shared Memory

RESERVED

7.6 Custom TIE Instructions

RESERVED

7.6.1 PSRAND TIE Instruction

RESERVED

7.6.2 IDIV TIE Instruction

RESERVED

7.6.3 Leaky Bucket TIE Instruction

RESERVED

7.7 Rx ATM Traffic Management

The Rx AIT Processor use the Dual Leaky Bucket Policer and Early Packet Discard/Partial Packet Discard to manage ATM traffic. One Leaky Bucket checks for sustain rate and the other Leaky Buck burst rate conformance. The Dual Leaky Buckets can set the CLP bit (in the Traffic Descriptor, extracted from the ATM cell header). A set CLP bit allows agents down stream to drop this cell if necessary.

7.7.1 ATM Packets

RESERVED

7.7.2 AAL5 Frames

RESERVED

7.7.3 Dual Leaky Bucket

RESERVED

7.7.4 ATM Traffic Policing

RESERVED

7.7.5 Early Packet Discard / Partial Packet Discard (ATM/AAL5 Packets)

RESERVED

7.7.6 Multi-Threshold

RESERVED.

7.7.7 Early Packet Discard

RESERVED

7.7.8 Partial Packet Discard

RESERVED

7.7.9 Tail Packet Discard, TPD

RESERVED.

8 Receive Data Link Manager & Memory Controller

The Receive Data Link Manager (RxDLM) manages the external RX memory which is used to store RX Data and RX Control Parameters. All of the ATM cells and all of the frame-based data received through the SONET interfaces and the UTOPIA interface are stored in the external RX memory while they wait to be scheduled to a Switch port. Parameters required by some of the RX data processing functions are stored in the external RX memory. The RX data processing functions using this external memory include: Traffic Policing, Congestion Management, Per-Flow Queue Management, QOS Queue Management, Free Queue Management and the Switch Arbiter and Scheduler.

The RxDLM interfaces to a Memory Controller. The Memory Controller takes read and write requests from the RxDLM and transforms them into physical memory cycles. The interface between the RxDLM and the Memory Controller is through FIFOs. The RxDLM pushes read requests and write requests with write data into these FIFOs. The Memory Controller reads from these FIFOs and performs the requested access to external memory. The interface between the Memory Controller and the RxDLM is very specific to the functionality required by the RxDLM. The Interface between the External Memory and the Memory Controller is very specific to the selected memory technology. As a design goal, the selected memory technology could be changed with no impact to the Rx DLM and impact to the Memory Controller only in the actual interface to the external memory.

The series of memory accesses required by the RxDLM is predetermined and is explained in detail in sub-sections of this section of the document. As shown in Figure 8-1, the Memory Controller provides a separate port for each of these predetermined external memory accesses. The Memory Controller queues all pending memory access requests in an order that uses the interface to the external memory in the most efficient way. Efficiency is measured as the percent of bus cycles that are used to transfer data versus the number of bus cycles left empty.

The Memory Controller provides two ports for the Host interface to read and write the external memory.

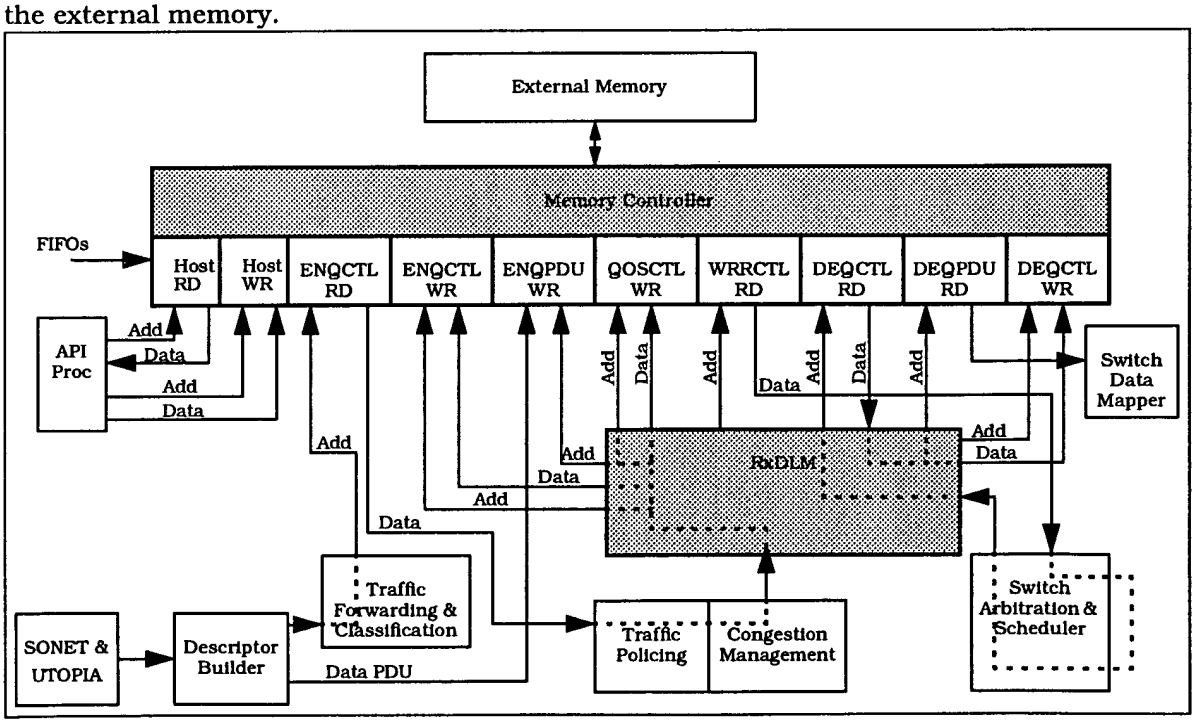


Figure 8-1: RxDLM Interface to Memory Controller

8.1 External Memory Structure

The External Memory Structure is shown in Figure 8-2. The external memory is divided into three sections: PDU Storage; Per-Flow Parameters; and QOS Request Queues.

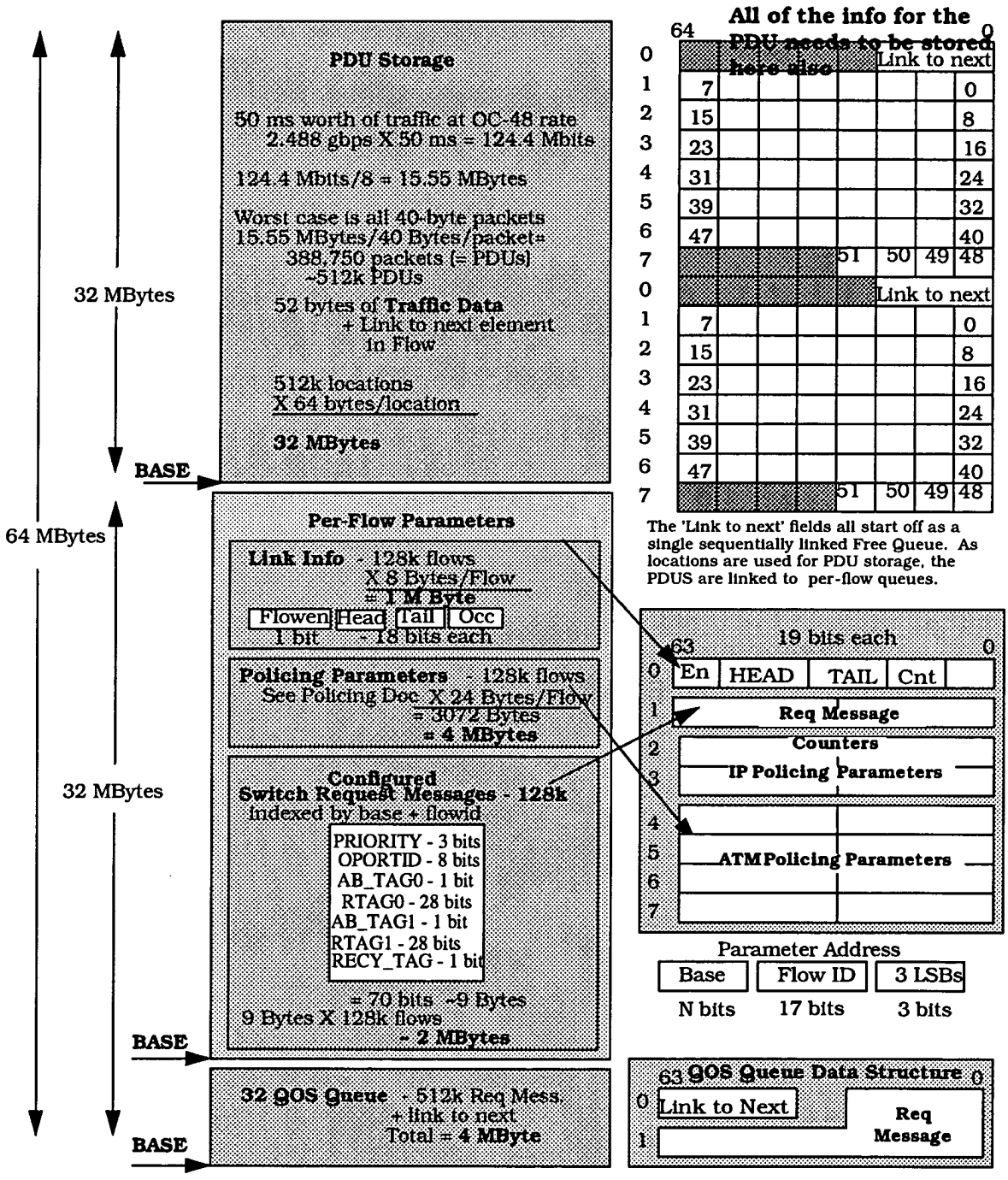


Figure 8-2: Rx External Memory

8.1.1 PDU Storage

The RxDLM stores 52-byte ATM Cells (no HEC) and 52-byte packet chunks in external memory. Each of these 52-byte chunks is stored in a 64-byte segment of memory. This 64-byte segment of memory is called a Payload Data Unit (PDU). There is no distinction here between ATM cells and IP packets. They are all stored and retrieved as 52-byte entities - cells or chunks. The Switch Scheduler schedules packet chunks before the entire packet is received, so the Rx_DLM does not need a packet building link manager.

The Receive side of the Service Processor is required to store a minimum of 50 ms worth of 40-byte IP Packets arriving at an OC-48 rate. The equation in Figure 8-2 shows that 512k PDUs of storage exceeds this requirement. The Service Processor can store PDUs in increments of 64 bytes. 512k PDUs at 64-bytes per PDU equates to 32 MBytes of storage.

A Linked-List data structure is used to manage the PDU Storage Memory. The Service Processor can support up to 128k different traffic flows and each one of these traffic flows is allocated its own set of Linked-List Parameters. These Linked-List Parameters consist of a Head, Tail and Count in units of PDUs. These Linked List Parameters are stored in the external memory with the Per-Flow Parameters (see Section 8.1.2).

The Count is stored on a per-flow basis for control purposes. If a particular flow or set of flows is filling the buffer to an unusually high level, then something is wrong. Maybe the output Port Processor is not scheduling it or the output flow is clogged... Whatever the reason, once the flow exceeds a defined threshold, the DLM will alert SW so that some corrective action can be taken. A global threshold can be set to limit the number of PDU locations that can be used by any single flow.

In order to support this Linked List memory management, a Free Queue needs to be maintained. Initially, all PDU locations belong to the Free Queue. As PDU elements arrive, storage locations are allocated from the HEAD of the Free Queue and appended to the TAIL of the per-flow linked-list. As PDU elements are scheduled out of the external memory, the storage locations are returned to the Free Queue by appending the location to the TAIL of the Free Queue. Since the "Link to Next" field of the stored PDU must be written when the PDU is written, a free PDU is pre-allocated to the TAIL of each of the active per-flow queues.

In order to minimize the number of accesses to external memory during memory intense conditions, a cache of Free Queue locations is kept internal to the Service Processor. The Free Queue cache is a list of pointers to a subset of the Free Queue. This Free Queue Cache is required for the worst case equilibrium state in which traffic is being queued at an OC-48 rate and traffic is being de-queued at the same OC-48 rate. In this equilibrium state all external memory cycles are used for enqueuing and dequeing PDUs and control information resulting in no memory cycles left over for Free Queue maintenance. In this worst case equilibrium state, as PDUs are dequeued, the PDU locations are internally returned to the Free Queue cache to be used immediately to enqueue the incoming PDUs.

In a non-equilibrium state, the rate of enqueues does not equal the rate of dequeues. If the rate of enqueues is greater than the rate of dequeues, then the unused PDU dequeue opportunities are used to replenish the internal Free Queue cache by reading Free Queue locations stored externally. If the rate of dequeues is greater than the rate of enqueues, then the unused PDU enqueue opportunities are used to append elements of the growing internal Free Queue cache to the external Free Queue.

The size of this Free Queue cache is TBD.

8.1.2 Per-Flow Parameters

The Per-Flow Parameters consist of all of the configuration parameters, link management information and performance monitoring that needs to be maintained separately for each of the 128k data flows. The data structure for the Per-Flow Parameters is shown in Figure 8-3. This data structure must be retrieved from external memory for every received ATM cell and for the first PDU accumulated for every frame or packet. There is no need to retrieve this information for any PDUs other than the first of each frame or packet because all algorithms and parameters are designed to operate in increments of packets and frames. This being the case, information describing multi-PDU packets and frames is kept internal to the Service Processor so that the yet to arrive PDUs will be

queued to the correct flow and QOS. One set of parameters per PHY port and SPE is kept internally as interleaving of packets and frames on the same PHY port or SPE is not allowed - except for the case of AAL5 in which the SOP/EOP delineation and the packet pass/drop status is kept externally with the Per-Flow parameters.

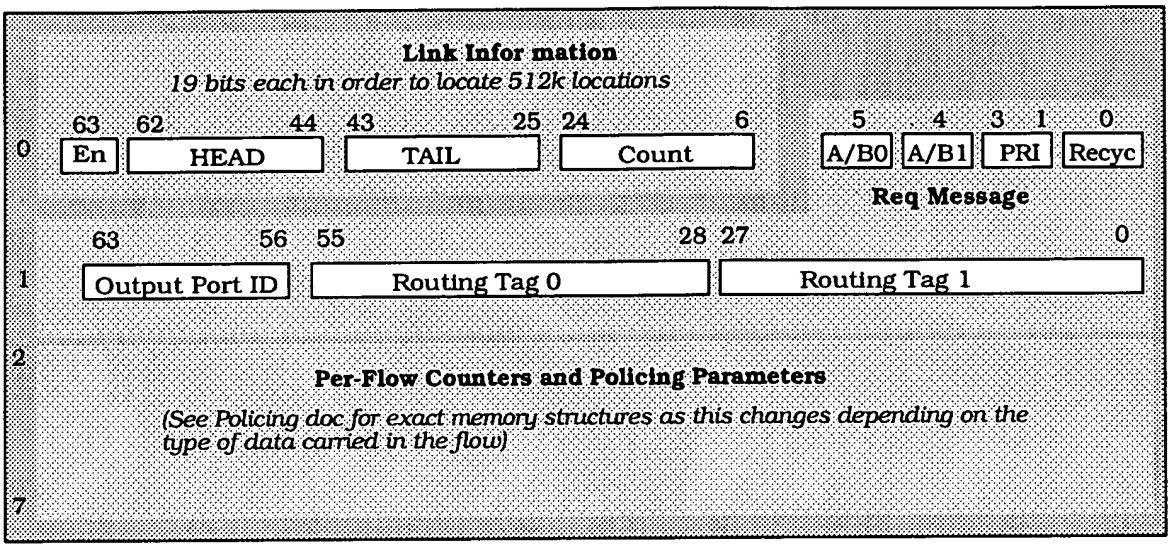


Figure 8-3: Per-Flow Control Data Structure

8.1.3 QOS Request Queues

Arriving PDUs must request access to and through the switch fabric. Once the PDUs are stored in PDU Storage, a request for the PDU must be queued in the QOS Request Queues. The switch scheduler will monitor these request queues and will map the requests into an outgoing message structure. The PDUs are classified into one of 32 different Classes or levels of QOS. These 32 classes dictate the priority of the PDU in the weighted scheduling algorithm that is used to schedule and arbitrate for routes through the switch fabric.

The information needed to build a switch request message and the link to the next element in the queue is stored in the QOS Request Queues. There are 32 queues - one for each class of service. The information needed to build a request and the structure of this data is shown in Figure 8-4.

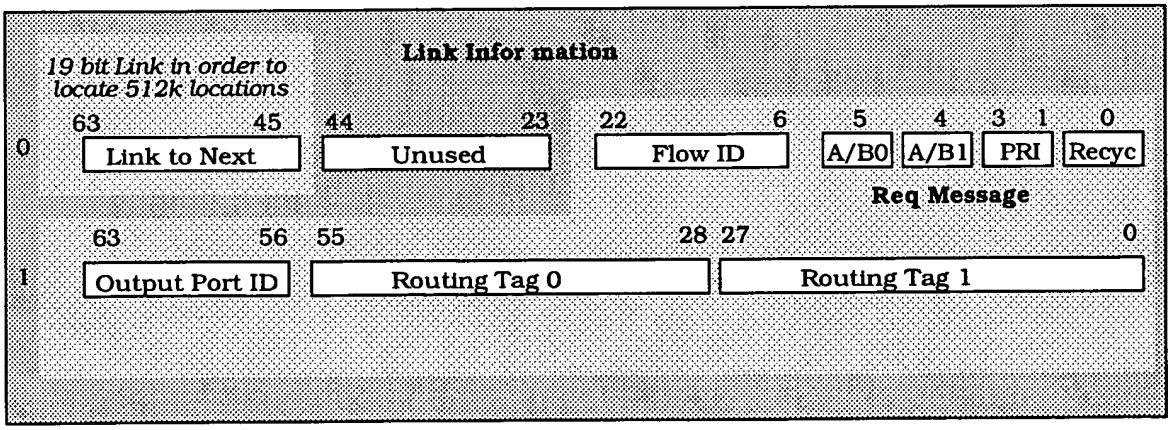


Figure 8-4: QOS Queue Data Structure

A linked-list is used to manage the 32 separate queues. Since there are only 32 queues, the

linked list parameters - Head, Tail and Count - are kept internal to the Service Processor. The elements in these queues are stored in order of PDU arrival. A separate entry is made to these queues for every PDU arrival. PDUs from different flows will be intermingled, but the cells and chunks in a particular flow will maintain order.

These QOS Queues are extended into the Service Processor. The tops of the queues are stored in 32 internal FIFOs. As long as there is room in these internal FIFOs, the Request elements will be pushed into these FIFOs. The RxDLM is responsible for keeping these internal FIFOs full. As these internal FIFOs become full, the request elements will be stored in the external QOS Request Queues.

8.2 Host RD

8.3 Host WR

8.4 Data & Control Flow

Figure 8-5 shows the required memory accesses and the order of the memory access that are required to enqueue and dequeue a single PDU. The accesses always follow the order shown in Figure 8-5. This prescribed order allows for the maximum utilization of the external memory bus. This sequence of memory accesses is designed to finish in less than the minimum sized packet (48 bytes) arrival time at an OC-48 rate when processing IP packets and in less than the arrival rate of ATM cells at an OC-48 rate when processing ATM traffic. To fully process a PDU it takes many packet/cell processing times. When cells or frames are received back-to-back, the memory cycles for each are pipelined as shown in Figure 8-6.

The high-level description of the PDU and Control flow in and out of external memory is in Figure 8-5. More detailed descriptions of these memory cycles is given in the following subsections.

The PDU has gone through the Descriptor Builder and the Traffic Forwarding. Based on the Flow ID from the Traffic Forwarding Block, retrieve the entire 64-byte control data structure for the Flow of the incoming PDU. This includes: the Linked List Control (= Head, Tail & Count); Req Message; per-flow counters; Policing Parameters for IP; Policing Parameters for ATM.

N X Packet Processing Times

The Policing information is added to the Descriptor before the Policing block. The PDU is in temporary storage in the RxDLM waiting for the Descriptor

The Descriptor has gone through the Policing function, through the Congestion Management function, and has arrived at the Rx DLM.

Based on the Pass/Drop indication, the PDU is enqueued to the external PDU Memory location pointed to by the TAIL field of the Linked List Control. The TAIL field of the Linked List Control is updated to the location pointed to by the HEAD of the internal FREE QUEUE cache.

The control data structure can now be written

Now that the PDU is queued in external memory, the request for the PDU needs to be queued to the correct QOS queue so that a request for the PDU can be sent to the SWIF.

The entire request message + Flow ID is queued to the external QOS queue location pointed to by the TAIL of the internal QOS Linked List control.

The TAIL of the internal QOS Control is updated with the HEAD of the internal QOS

The PDU is in external memory. The Flow Parameters are updated. The PDU REQ is queued to an external QOS queue.
Now need to get the req to the internal QOS queue so that it can be arbited and then granted admission to the SWIF.

The Request Messages for each QOS Queue need to be transferred from the large external queues to the smaller internal queues. These smaller internal queues are used by the WRR Scheduler.

The WRR Scheduler will eventually send to the Rx DLM either the FLOW ID to be dequeued OR the entire request message to be recycled. Recycling involves requeueing the Request Message to the external QOS Queue which means borrowing a "WR QOS CTL" cycle.

Read the Linked List Control for the Flow ID selected for Dequeue. Need to Dequeue PDUs from the location pointed to by the HEAD.

Update the HEAD field with the "link to next" from the PDU

Read the PDU in the location pointed to by the HEAD of the DEQ CTL.

Update the Flow HEAD field with the "link to next" from the retrieved PDU.

Decrement the Count.

Return the PDU Location to the internal FREE QUEUE cache.

Write back the updated Flow parameters.

Figure 8-5: External Memory Access Flow

	RD ENQ CTL	RD DEQ PDU	RD DEQ CTL	RD WRR CTL	WR ENQ CTL	WR ENQ PDU	WR DEQ CTL	WR QOS CTL

One Packet/Cell Processing Time

	RD ENQ CTL							
P	RD ENQ CTL							
CM	RD ENQ CTL							
	RD ENQ CTL				WR ENQ CTL	WR ENQ PDU		WR QOS CTL
	RD ENQ CTL			RD WRR CTL	WR ENQ CTL	WR ENQ PDU		WR QOS CTL
	RD ENQ CTL		RD DEQ CTL	RD WRR CTL	WR ENQ CTL	WR ENQ PDU		WR QOS CTL
	RD ENQ CTL	RD DEQ PDU	RD DEQ CTL	RD WRR CTL	WR ENQ CTL	WR ENQ PDU	WR DEQ CTL	WR QOS CTL
	RD ENQ CTL	RD DEQ PDU	RD DEQ CTL	RD WRR CTL	WR ENQ CTL	WR ENQ PDU	WR DEQ CTL	WR QOS CTL
	RD ENQ CTL	RD DEQ PDU	RD DEQ CTL	RD WRR CTL	WR ENQ CTL	WR ENQ PDU	WR DEQ CTL	WR QOS CTL
	RD ENQ CTL	RD DEQ PDU	RD DEQ CTL	RD WRR CTL	WR ENQ CTL	WR ENQ PDU	WR DEQ CTL	WR QOS CTL
	RD ENQ CTL	RD DEQ PDU	RD DEQ CTL	RD WRR CTL	WR ENQ CTL	WR ENQ PDU	WR DEQ CTL	WR QOS CTL
		RD DEQ PDU	RD DEQ CTL	RD WRR CTL	WR ENQ CTL	WR ENQ PDU	WR DEQ CTL	WR QOS CTL
		RD DEQ PDU	RD DEQ CTL	RD WRR CTL	WR ENQ CTL	WR ENQ PDU	WR DEQ CTL	WR QOS CTL
		RD DEQ PDU	RD DEQ CTL	RD WRR CTL	WR ENQ CTL	WR ENQ PDU	WR DEQ CTL	WR QOS CTL
		RD DEQ PDU	RD DEQ CTL	RD WRR CTL			WR DEQ CTL	
		RD DEQ PDU	RD DEQ CTL				WR DEQ CTL	
		RD DEQ PDU					WR DEQ CTL	
	7	1	2	3	4	4	1	4
	T0	T-6	T-5	T-4	T-3	T-3	T-6	T-3
EnQ	T0				T-3	T-3		T-3

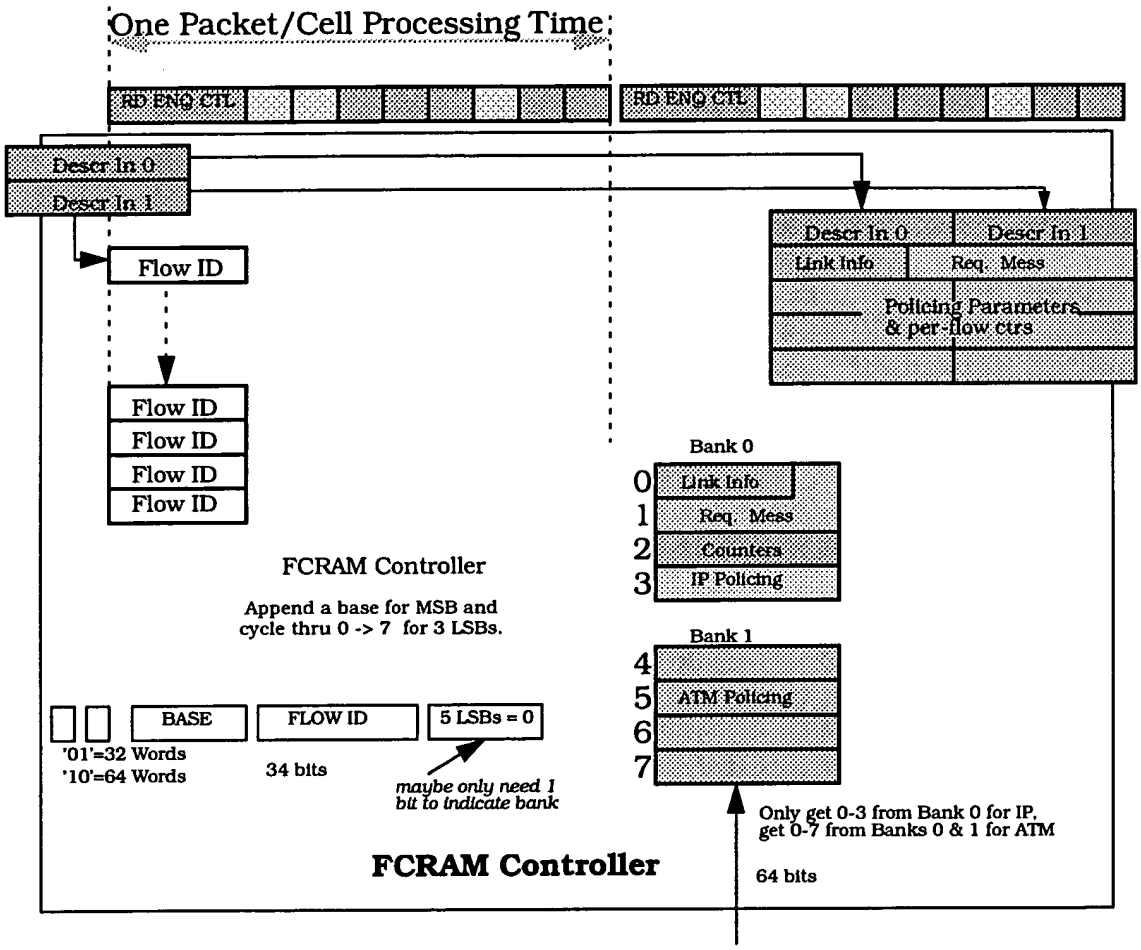
From the time that the ENQ Control Data Structure is read out of external memory, 4 packet/cell processing times go by before this control information is written back to external memory. This means two things:

1. The Rx DLM must be able to store at least 4 ENQ Control Data Structures.
2. If consecutive packets or cells from the same flow arrive within 4 packet or cell processing times of each other, then the ENQ Control Data Structure must be read from the processors internal cache as the external data structure will be holding stale information.

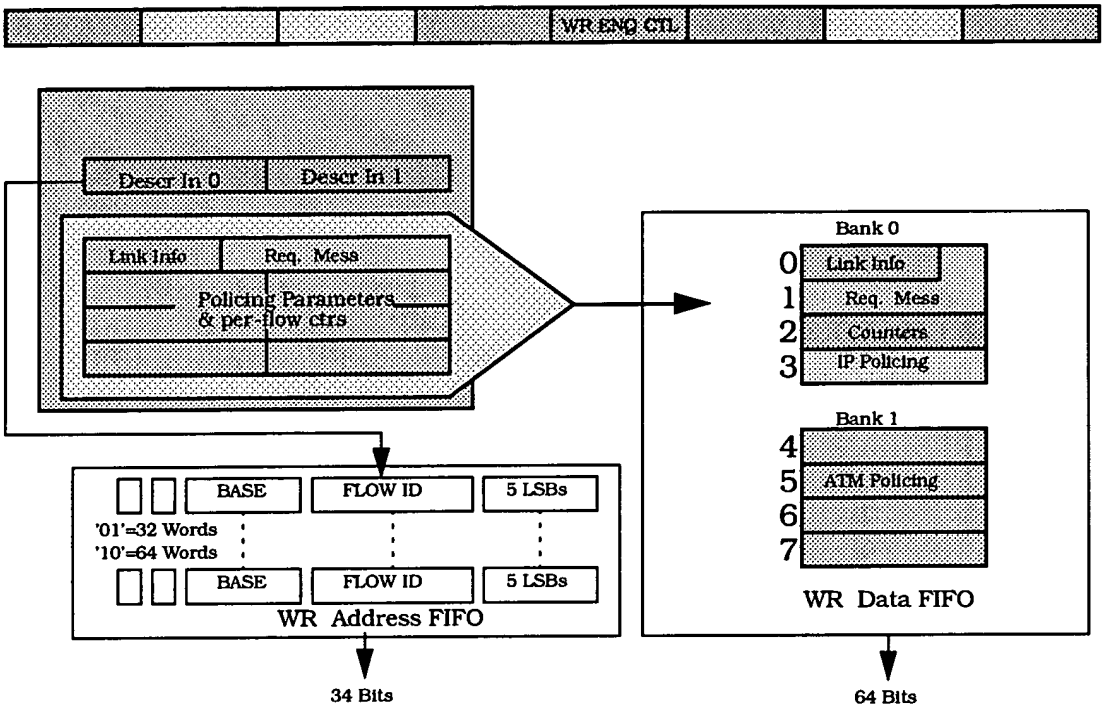
Figure 8-6: External Memory Access Pipeline

8.4.1 RD ENQCTL

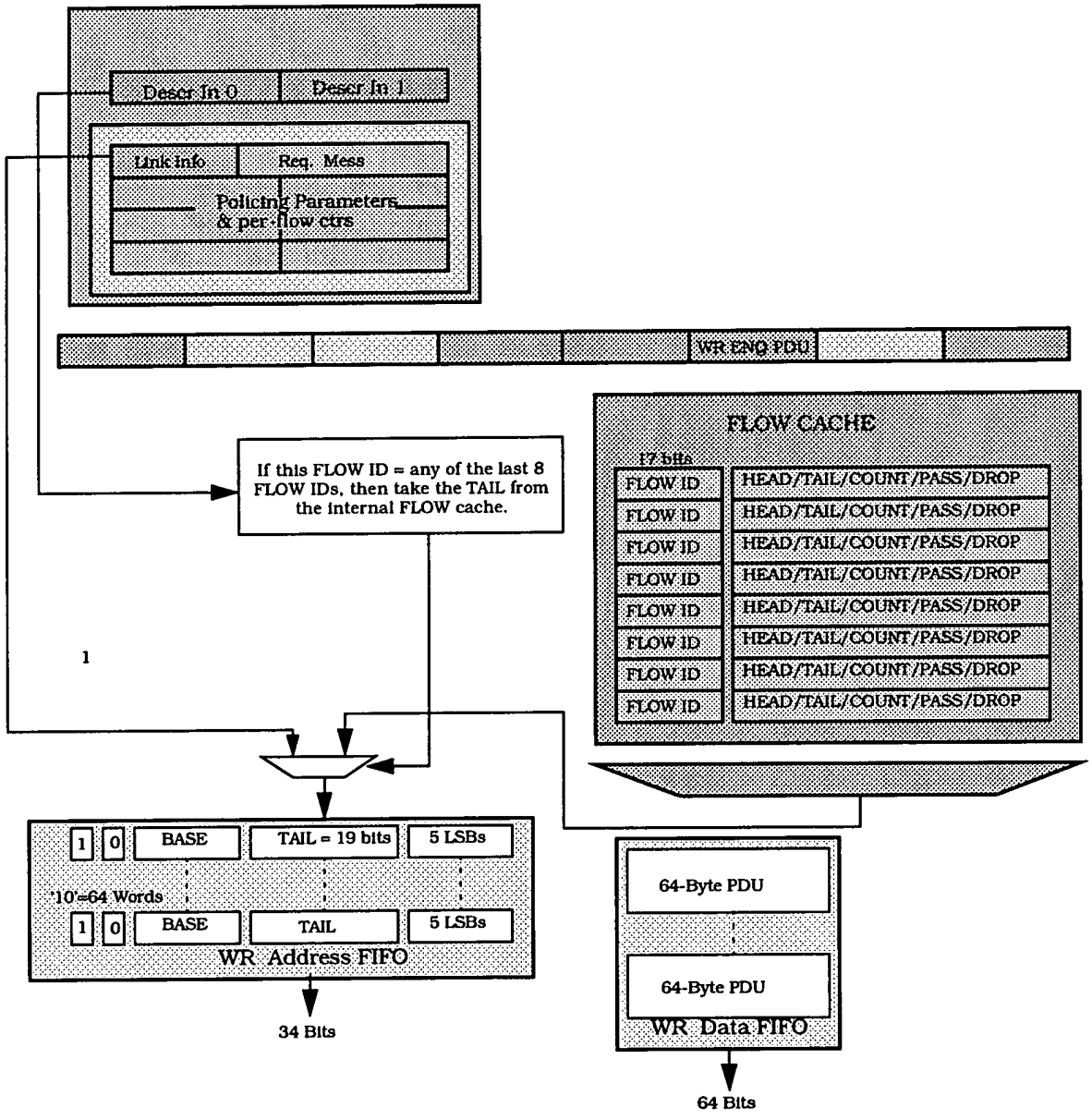
Read the Enqueue Control Data Structure.



8.4.2 WR ENQCTL - Write the Updated Enqueue Control Data Structure



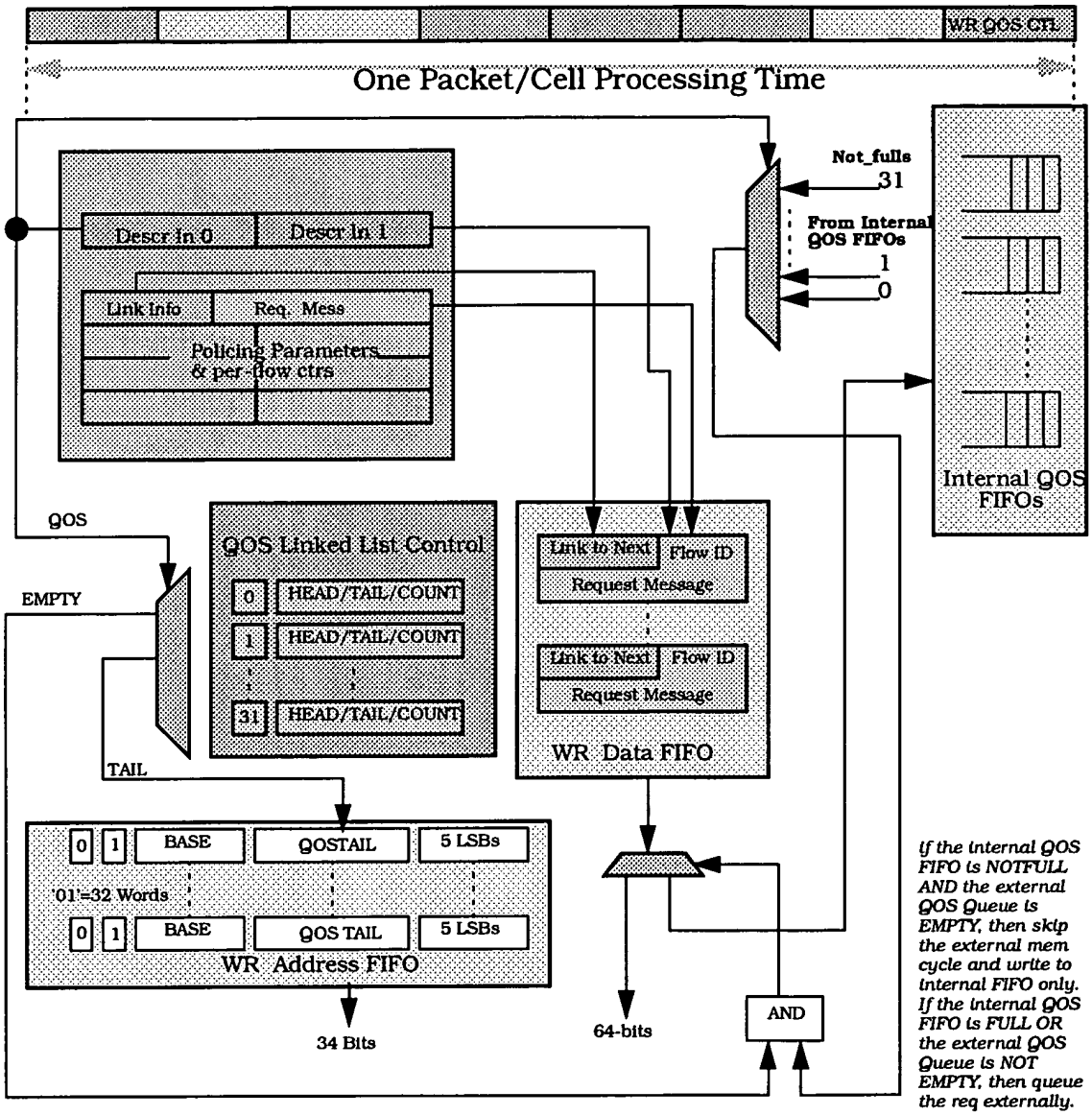
8.4.3 WR ENQPDU - Enqueue a Data PDU



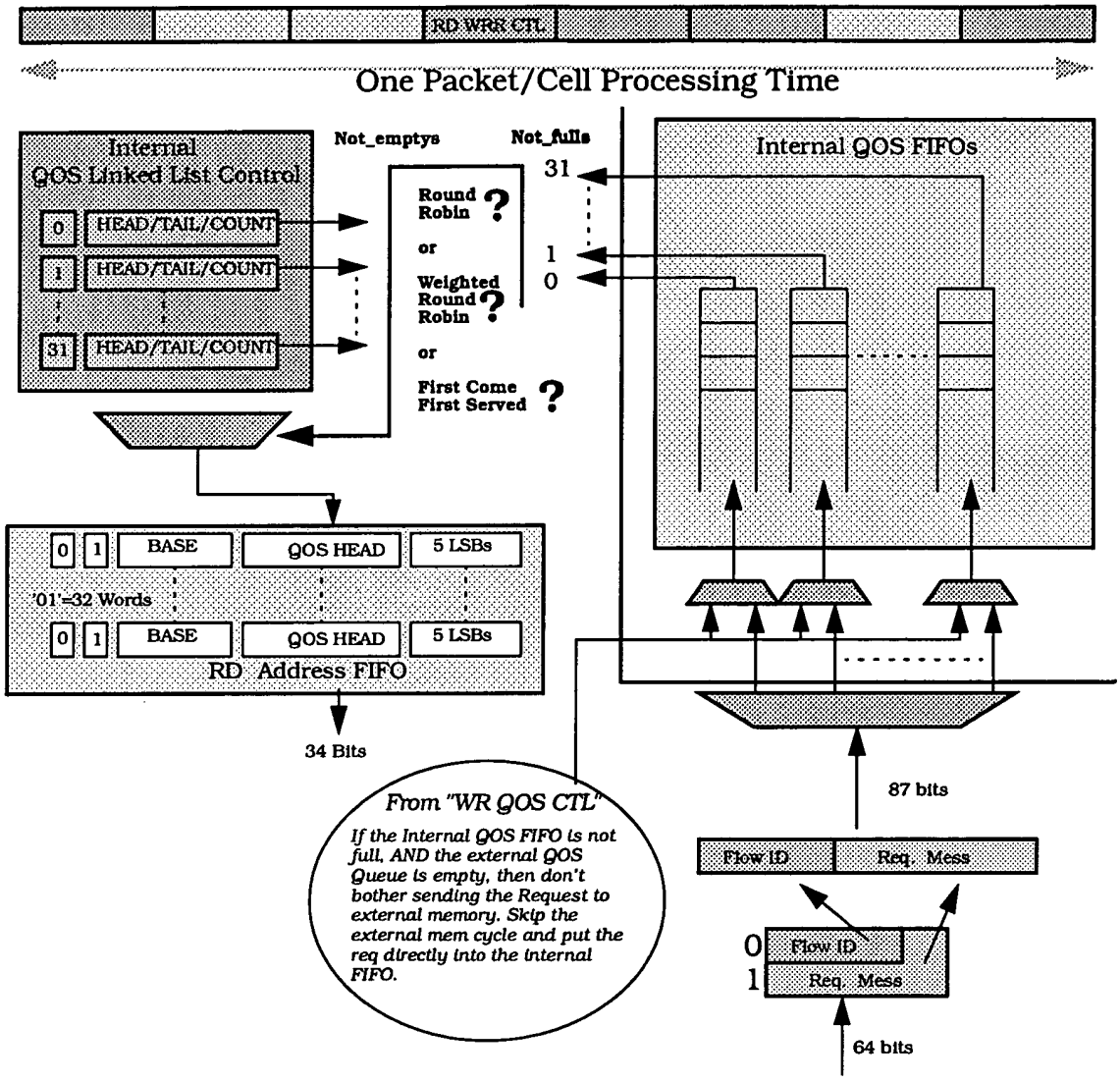
When the Data FIFO from the Descriptor Builder indicates 'not empty', the DLM must remove a 64-byte element from the FIFO. The DLM writes the cell or chunk to the location pointed to by the head of the free queue. If this is an ATM cell or the first chunk of a packet (SOP=1), then the DLM waits for the descriptor to arrive so that it can identify to which flow queue to enqueue the data and also to which QOS Queue to enqueue the request. For SOP packet chunks, the DLM writes the Flow ID and the QOS ID into a dedicated location to be referenced when the rest of the packet arrives. The PDU counter is updated on every PDU arrival. A Data PDU is always 64-bytes. The PDU can contain either a 52-byte ATM Cell or a packet chunk that is less than or equal to 52 bytes in length. The packet counter is only updated on the arrival of the first chunk of an IP packet. The packet counter is not updated for ATM cell arrival. The packet chunks that arrive after the first chunk will have a descriptor accompanying them and the SOP bit will = 0. This tells the DLM to check the non-SOP

RAM to find the flow ID and the QOS level.

8.4.4 WR QOS CTL



8.4.5 RD WRR CTL



8.5 Element Arrival

8.5.1 ATM CELL

Queue the PDU and when the descriptor arrives, link the PDU to the correct Flow and QOS queues.

1. Write PDU to the location pointed to by the Head of the Free Queue. - **7 writes.** Keep the pointer as it will need to be written into the "link to next" field of the last item in the flow queue or to the Head of the flow queue.

The top portion of the Free Queue is stored internally and is updated as a background task. This background task will need to be quantified.

When the Descriptor arrives it will contain the Flow ID and a QOS.

2. Get the Head & Tail & Occupied and Thresh of the Flow ID Queue - **2 reads** from the parameters
3. a. If Head and Tail are not equal, Write the pointer to where the PDU was just written to into the "link to next" field of the last written (before this one) PDU. - **1 write**

   b. If Head =Tail, then write the pointer into the Head value for the flow - **1 write**

4. Get the Head & Tail of the QOS queue. **1 read**
5. a. If Head and Tail are not equal, Write the pointer to where the PDU was just written to into the "link to next" field of the last written (before this one) link. - **1 write**

   b. If Head =Tail, then write the pointer into the Head value for the QOS queue - **0 write** *All QOS heads and tails should be stored in internal memory*

6.

9 Switch Controller

The switch controller is the control interface between the port processor and the switch. The functionality of the switch controller can be logically broken down into two sections (see 9-1).

The first switch controller section (RX Switch Controller) interfaces the receive direction of the switch data-path and the switch. The second switch controller section (TX Switch Controller) interfaces the transmit direction of the switch data path and the switch. These two switch controllers are described in detail in the following sections.

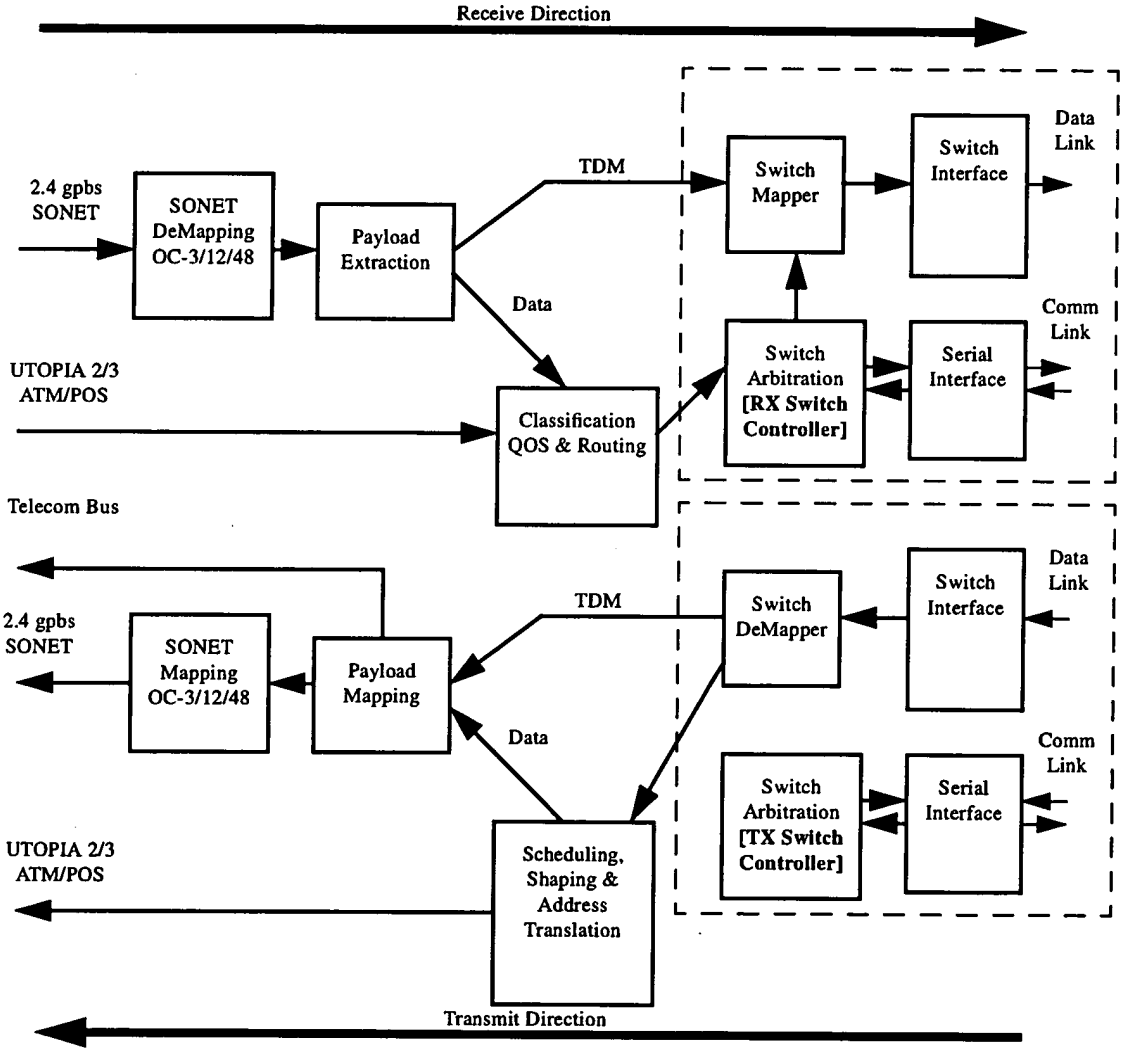


Figure 9-1: iTAP Port Processor Overview

9.1 RX Switch Controller

The TDM traffic is circuit switched and circuits are provisioned through the switch for the TDM traffic. The RX Switch Controller does not handle any control signals for the TDM traffic. The bandwidth transmission capacity remaining after provisioning of the TDM traffic is available for transmission of IP and ATM Traffic. The TDM and ATM traffic is packet switched through the switch fabric in fixed size cell containers of 64 bytes including overhead. The format of a cell container is documented in the overview section of the ITAP switch chip engineering specification. (An alternative term used for these cell containers is Payload Data Unit or PDU's).

9.2 RX Transmission Path

Each incoming cell and packet is processed by the RX port processor. In order to perform this algorithmic processing, the Port Processor needs to retrieve the set of configured parameters for each ATM cell or PPP frame. For ATM Cells the Port Processor performs a lookup based on the ATM VPI/VCI. This lookup will first verify that the connection is active and if active, it will return an 17-bit index to a set of per VC parameters and to routing information. The 17 bit index supports a maximum of 128K simultaneous IP and ATM flows through the port processor.

IP Packets are forwarded to an internal processor allocated to performing routing based on a third party Longest Prefix Match algorithm. The result of the lookup process is verification that the connection is active, and if active the lookup will return an 17-bit index to a set of queueing parameters and to routing information.

The ATM cells are encapsulated in a cell container and stored in one of 128K queues in external memory. These 128K queues are managed by the Data Link Manager which is not part of the RX switch controller. Control information required to transmit the packet is forwarded to the RX Switch Controller.

The IP packets are fragmented into 51 byte blocks and each of these blocks are encapsulated in a cell container. These cell containers are stored in one of 128K queues in external memory by the data link manager and control information required to transmit the packets is forwarded to the RX switch controller. A conceptual overview of the RX switch controller is shown in 9-2.

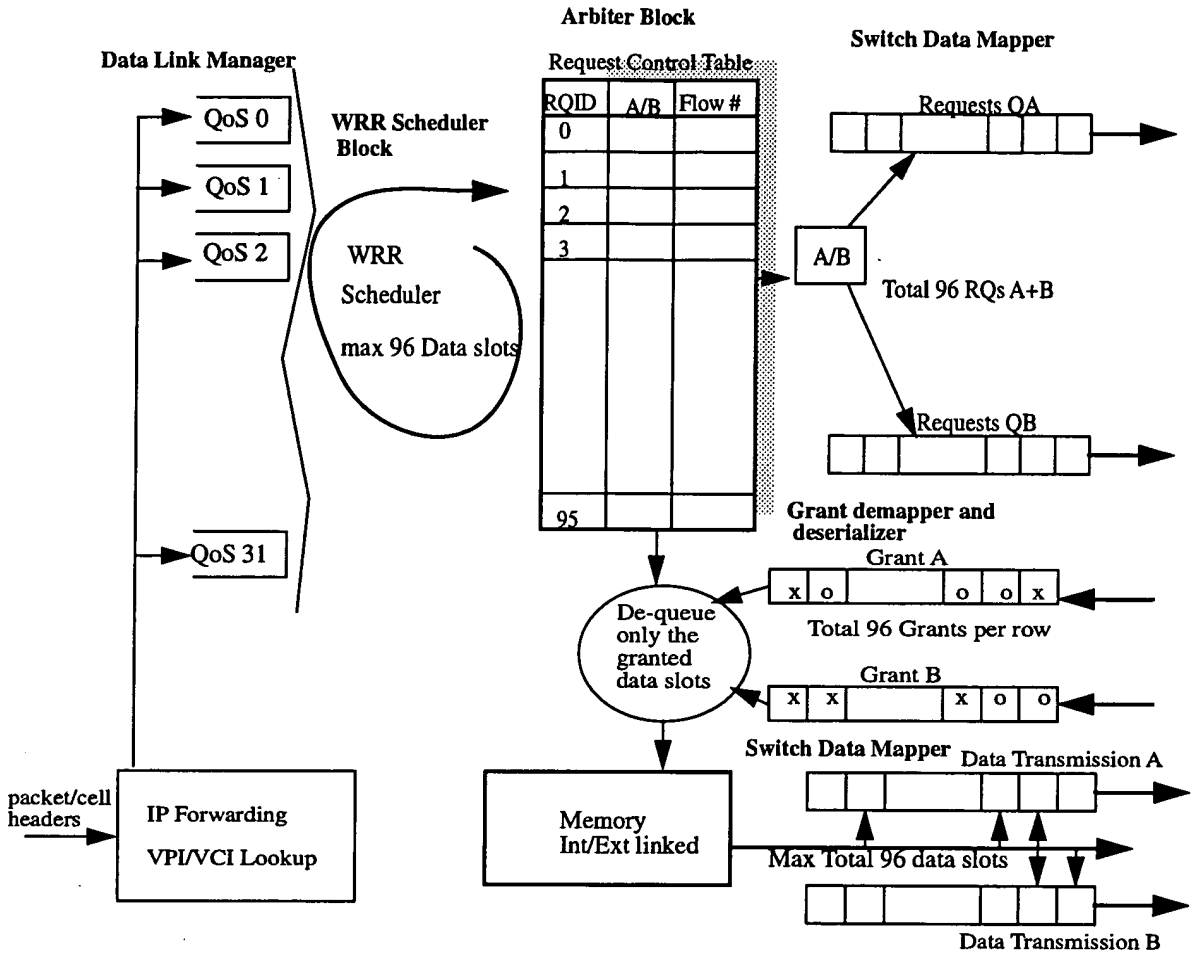


Figure 9-2: Overview of the RX Switch Controller

The 128K IP/ATM flows are aggregated into one of 31 QoS queues for scheduling through the switch. The data link manager aggregates all the control headers required for transmission of cells through the switch into 31 QOS queues and inserts these routing tags into one of 31 QOS routing tag fifos. One of the queues, here called QOS 0 is reserved for high priority traffic. Any cells arriving in the high priority queue will interrupt the scheduler and be scheduled to leave the high priority queue immediately.

The scheduler is responsible for scheduling cell containers through the switch. The scheduling algorithm used is Weighted Round Robin and operates on these 31 QoS queues. Once cells have been scheduled from these queues, the control headers from these queues are forwarded to the arbiter and are stored in a request control table.

The request arbiter forms requests from the control headers and forwards these requests to the switch data mapper block for transmission through the switch. The grants received in response to these requests are deserialized and deframed and transferred back to the arbiter block. For requests accepted for transmission through the switch, the cell containers are dequed from external memory by the data link manager and transferred to the switch data mapper for transmission through the switch.

A architectural overview of the RX switch controller with the data flow between blocks is shown overleaf in Figure Figure 9-3.

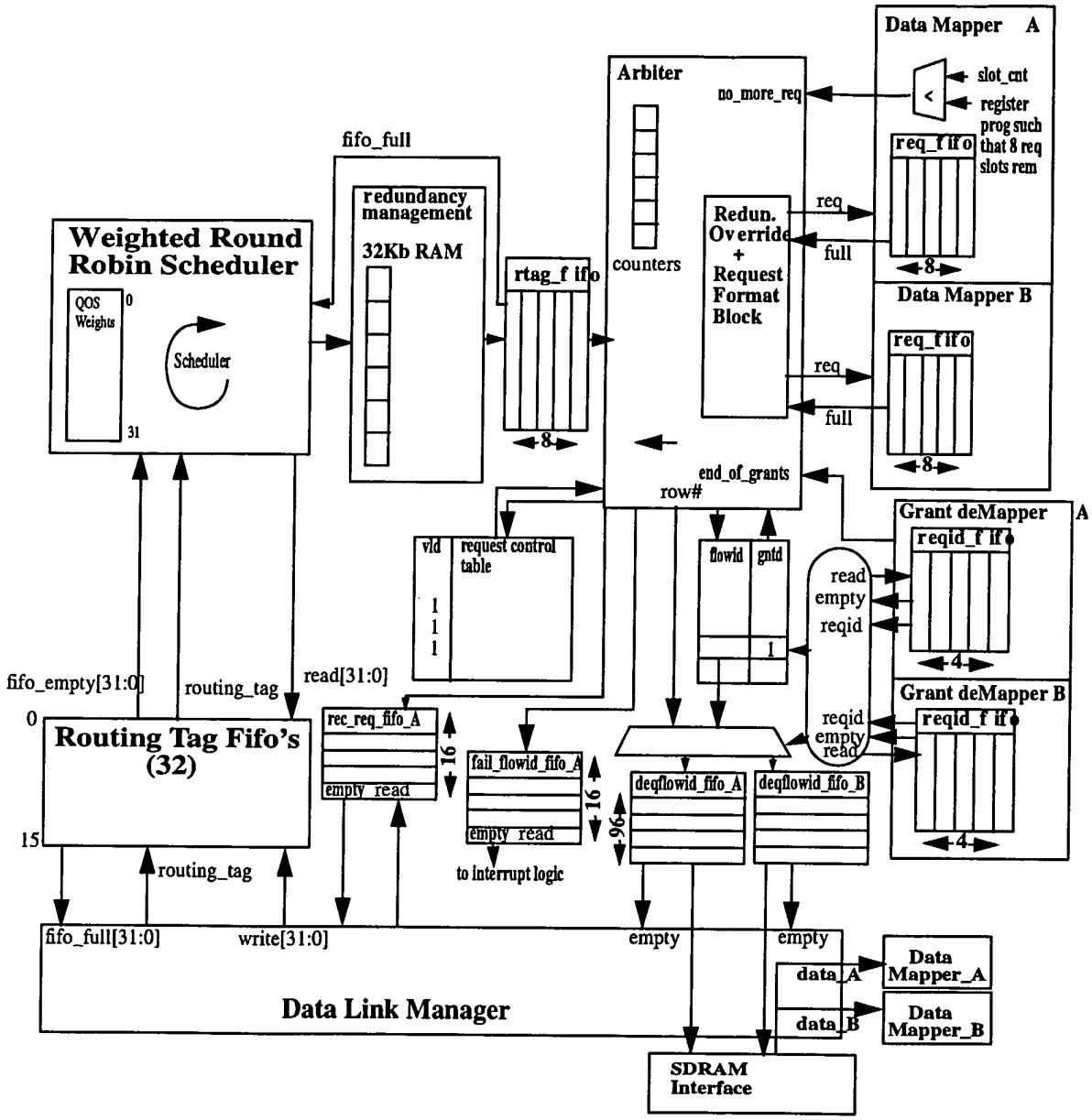


Figure 9-3: Data Flow between blocks in the RX Switch Controller

9.2.1 Routing Tag Fifo's

The Routing Tag Fifo's are 32 fifo's containing 16 routing tags each. The weighted round robin scheduler will schedule routing tags out of these 16 fifos and will monitor the 32 fifo_empty lines while scheduling. The Data Link Manager will monitor the 32 fifo_full lines and update the routing tag fifo's to keep them full.

9.2.2 Weighted Round Robin Scheduler

The Weighted Round Robin Scheduler selects routing tags from one of 31 QoS queues and schedules these routing tag for transmission through the switch. The weighted round robin scheduler is run each time there is a location in the rtag_fifo. One of the fifo's, fifo 0 will be given absolute highest priority, the arrival of routing tags in this fifo queue will interrupt the weighted round robin scheduling of packets, and these routing tags will be scheduled to leave the switch immediately. When there are no more routing tags in this queue, weighted round robin scheduling of routing tags from the remaining 31 queues will continue.

This implementation uses 1 counter - 8 bits wide.

```
if (fifo_empty[0]) /* highest priority */
  begin /* scheduling cycle */
  for (qos_no = 0; qos_no < 30; qos_no++)
    {current_queue = weight[qos_no]; /* weights are programmed at setup time */
      while (current_queue > 0)
      {insert_into_table (get_packet_from_head(qos_no));
       current_queue--;
   }
}
 end /* scheduling cycle */
```

9.2.3 Redundancy Management Block

In order to improve reliability, two redundancy schemes are supported. In the first redundancy scheme, the switch controller supports redundant routing tags and transparent route switch-over. In the second redundancy scheme, the port processor supports redundant data channels in both input and output directions. The redundant data channels connect to two separate switch fabrics. In this case we will call them A and B data channels.

Each control header will contain two routing tags, and each routing tag will have a corresponding AB channel tag. This provides for two routes through the switch for data transmission. If both routing tags have the same channel tag, this allows for two alternate paths through the same switch fabric. If both routing tags have different channel tags, this implies that there is a redundant switch fabric and any route failing in one switch fabric will cause a switch-over to use the redundant switch fabric.

Upon entry into the redundancy management block, a memory lookup using the most significant 15 bits of the flowid will be used to determine if the first or second routing tag is to be used. (A 32K bit memory is used in the redundancy management block to decide if the first or second routing tag should be used. Using 32K bits implies that we have aggregated four flows for the purpose of redundancy management and switch-over). The AB channel tag will indicate whether the data is to be routed using the A or the B data channel.

A request message will be sent using the appropriate routing tag for a programmable number of times. If no grant is received using the this routing tag, a bit will be set in the 16KB memory to switch over the routing tag for subsequent requests and the current request will be sent using the other routing tag. Setting the switchover bit implies that all four flows that share the same 14 msb bits in the flowid will be switched over as a group to use the other routing tag. The aggregation of flows into groups of 8 was done to save on chip memory.

9.2.4 Arbiter Block

The arbiter is responsible for sending requests to the data-mapper and processing the grants that arrive from the grant demapper. Initially the arbiter will deque requests from the rtag_fifo and

1. copy this information into the request control table,
2. write the flowid into the flowid ram

3. reset the request trial counter that counts the number of times a request has been tried
4. reset the grant bit.

The request message sent will have a unique request id (reqid) attached to the request, which is returned in the grant message. The reqid is the index in the arbiter request control table into which the routing tag is copied. The routing tag along with the reqid is forwarded to the routing tag formatter block which formats the routing tag into a request message and inserts the request into the request_fifo in the data-mapper block.

In the grant demapper block, the request id returned with the grant are stored in a fifo called the grant_reqid fifo. In the arbiter block, these reqids will be dequed from the A and B grant_reqid fifos alternatively. The reqids dequed from the fifo are used to

1. set a grant bit in the grant register at the bit position indicated by the request id.
2. index the flowid ram and read the flowid associated with the reqid. This flowid is written into the deq-flowid fifo for the appropriate channel, i.e if the requestid is dequeued from the A reqid_fifo, the flowid is written into the A deqflowid_fifo.

The data link manager monitors the deqflowid fifo and uses the flowid to deque data pdus from external memory and send them to the data mapper for transmission in the next row time.

The end_of_grants signal is asserted by the grant demapper, when no more grants can be received at the grant demapper. Once the end_of_grant signal has been received the arbiter begins the process of updating the request control table. If a grant has not been returned for a routing tag stored in the request control table, the request trial counter will be incremented and a new request will be generated using the routing tag.

If a routing tag in the request control table has been tried a maximum number of times (this number is programmed from 0 to 15), the most significant 14 bits of the flowid (out of the 17 bit flowid field) are used to index into the redundancy control table and update the bit to indicate failure of the current path and to select the alternate routing path. The current request will then be retried using the secondary routing tag for the maximum number of times.

If a packet could not be scheduled using either the primary or secondary routing tags, the request is removed from the arbiter request control table and the request is placed into a recycled_request fifo. A new request is dequeued from the rtag_fifo and copied into the request control table.

If the request has been granted a new request is dequeued from the rtag_fifo and copied into the request control table.

9.2.5 Recyling Requests

If a request could not be sent using either the primary or secondary routing tag, the request is removed from the request control table and the request is placed in the recycle request fifo. The removal of the routing tag from the request control table is to avoid head of line blocking. However, the request may not have been granted due to congestion at the output port and should be retried. (Congestion in the output port is the most likely scenario to be encountered by a request that has to be recycled, since congestion through the swich fabric is avoided by using the second routing tag). The recycled request is removed from the recycled request queue and written into the tail of the qos queue in external memory by the data link manager. There are two options that are supported in queing recycled requests. In the first option, the request is re-queued in to the original qos queue which contains packets for the flowid. i.e the qos of this packet is not changed in going through recycling. In the second option to be supported, a special qos number and queue is reserved for all recycled requests.

A recycled request is treated no differently in the arbitration block. A request message will be sent for this request and the request will be tried a programmable number of times using the first routing tag, if a grant is not recieved in the programmable number of times, the second routing tag will be tried next a programmable number of times.

Each time a request fails to be acknowledged with a grant, and it placed in the recycle fifo, the flowid is saved in the failed flowid fifo and an interrupt is made to the host processor along with the failing flowid.

The host processor will upon receipt of an error message indicating the inability to route the particular flowid perform diagnosis. Corrective action that can be taken is to either change the route by updating the routing tag in external memory, or shut off the flow by setting a bit in the routing tag to indicate an invalid routing tag. The data link manager will not queue routing tags for a particular flowid in which the invalid routing tag bit has been set.

9.3 TX switch controller

On the TX side of the port processor, the TX switch controller is responsible for either accepting or rejecting requests that come into the port processor for output transmission. In this case the TX switch controller has to check if the queue identified by the output port number of the request can accept a cell container. These 144 queues are stored in external memory and managed by the TX data link manager. The scheduling of these packets at the output is done by the TX scheduler. If the queue can accept the cell container, the request is turned into a grant and inserted into the grant_fifo. The grant-framer and serializer reads this information and creates an grant message for transmission through the grant path.

A data flow diagram and the data structures used in by the TX switch controller are shown below in 9-4

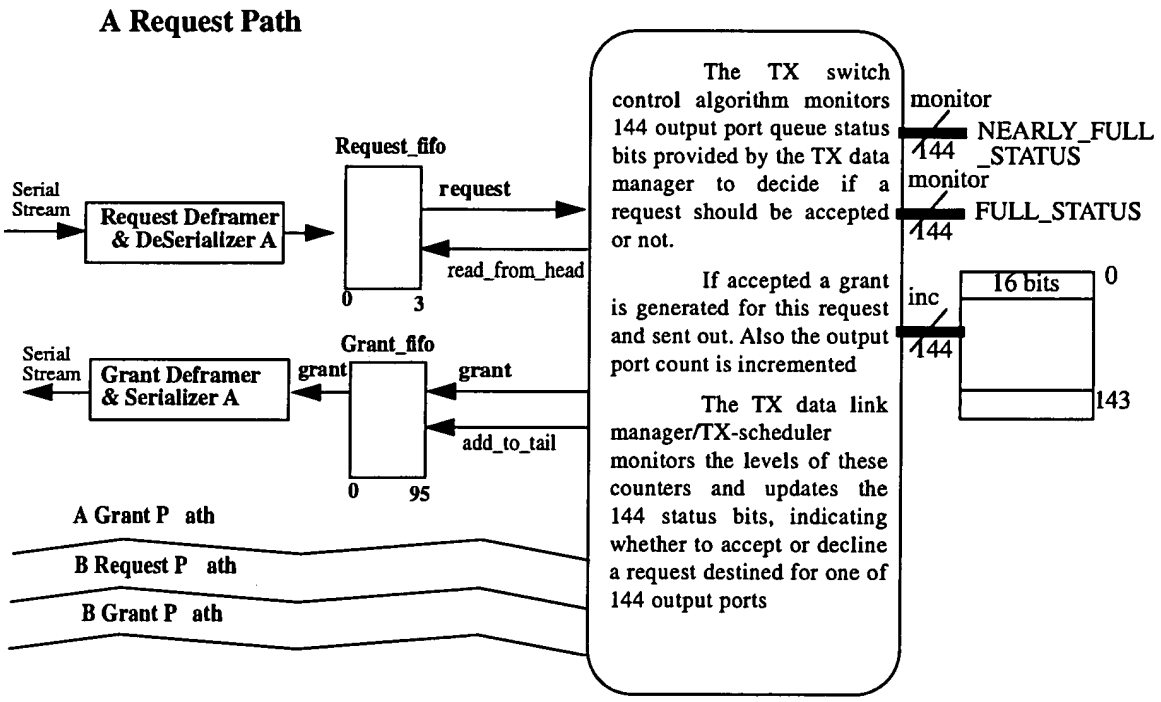


Figure 9-4: Data path and data structures in the TX switch controller

The acceptance of requests by the TX switch controller is done by monitoring the status of the data queues for each of the 144 output ports and using the following three rules

1. If the full_status bit for the request output port is set, there is no buffer space in the queue for any data pdus destined for that output port and all requests to that output port are denied.
2. If the full_status bit is not set and the nearly_full_status bit is set, this implies that there is some space in the queue for data pdus destined for that output port, however this space may be reserved for higher priority traffic. In this instance the QoS number is checked against a threshold programmed QoS and if the QoS number is less than the threshold, the request will be accepted, else it will be denied. (This implies that lower QoS numbers have higher priority)
3. If the nearly full_status bit is not set, all incoming requests are granted.

If a request is accepted the corresponding output port counter is incremented. This reserves space in the data buffer for the arrival of the data pdu at that output port. The transmit data link manager constantly monitors the 144 output port counters and sets/resets the 144 full and nearly full status bits.

10 Data Mapper

The TDM data, ATM/IP PDU's and the request messages are combined into a single data stream for transmission through 2 Serial links. (For more information read the overview section of the ITAP switch chip engineering specification)

This mapping is performed by the Data Mapper. A block diagram of the data mapper that maps the Row Buffer which contains the TDM and ATM/IP data and the requests and interfaces with the serializer logic is below in 10-1.

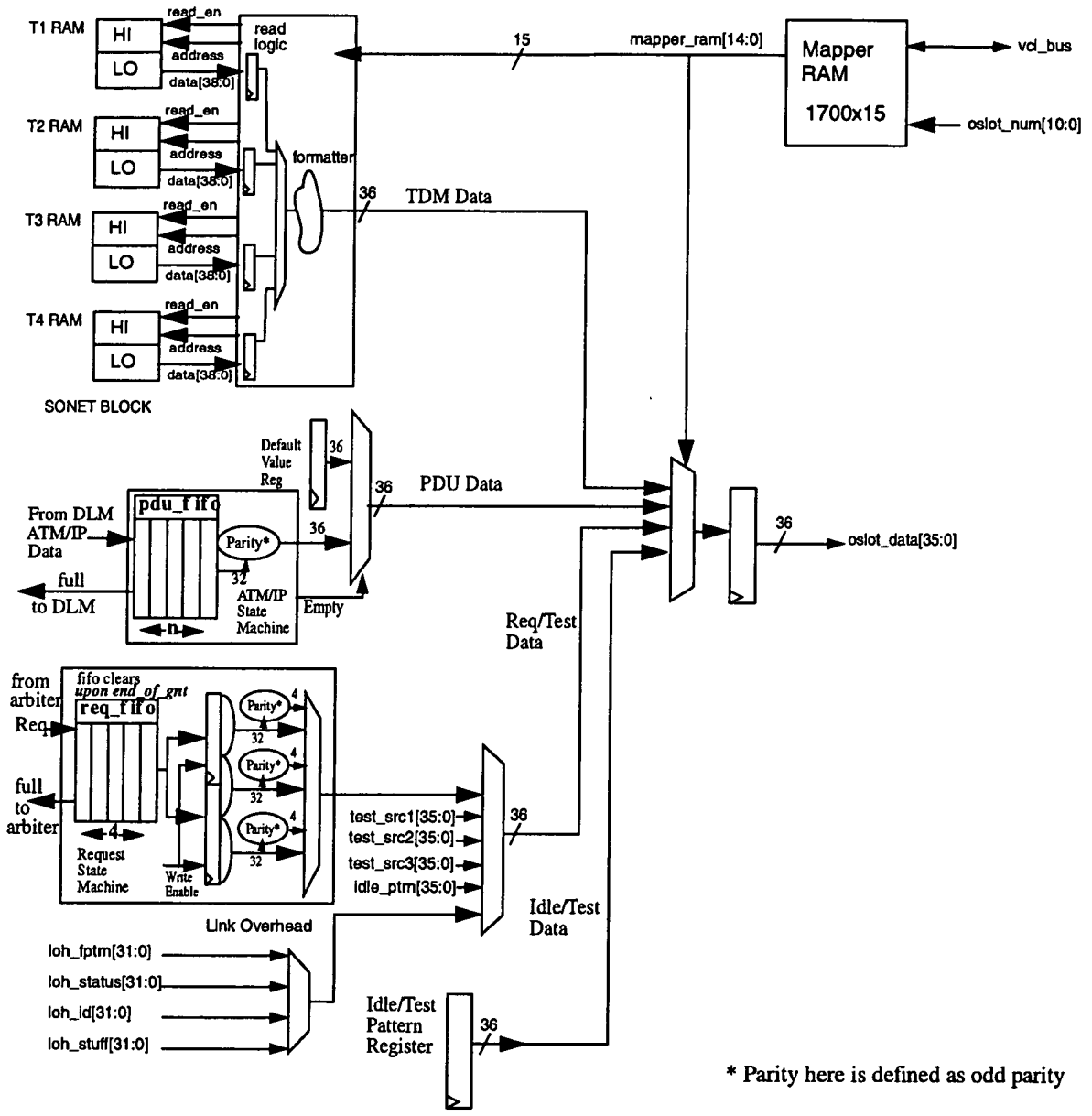


Figure 10-1: Request and Row Buffer Mapping

10.1 Mapper RAM

The Mapper RAM determines what data to map onto each slot of the output link.

The oslot_num input is incremented once for each outgoing slot and will be used as the address for this RAM. This output slot number will start at 0 for the first slot and then simply be incremented once for each new output slot up to the maximum number of slots in the row.

Since there will always be at least 2 cclk cycles per output slot, the accessing of the RAM is split into two phases. During phase 0 the RAM will be addressed using the oslot_num input, the output of the RAM will be registered at the end of phase 0 so that it will remain valid for the following 2 cclk cycles. During phase 1, the RAM may be written to or read from by the host processor via the vci bus. The RAM will be implemented with a 1 write, 1 read port memory macro.

The Mapper RAM is 15-bits wide. The first bits identify the type of data being transmitted, the value of these two bit determine how the next 13 bits are interpreted. The structure of the RAM is shown in Figure Figure 10-2

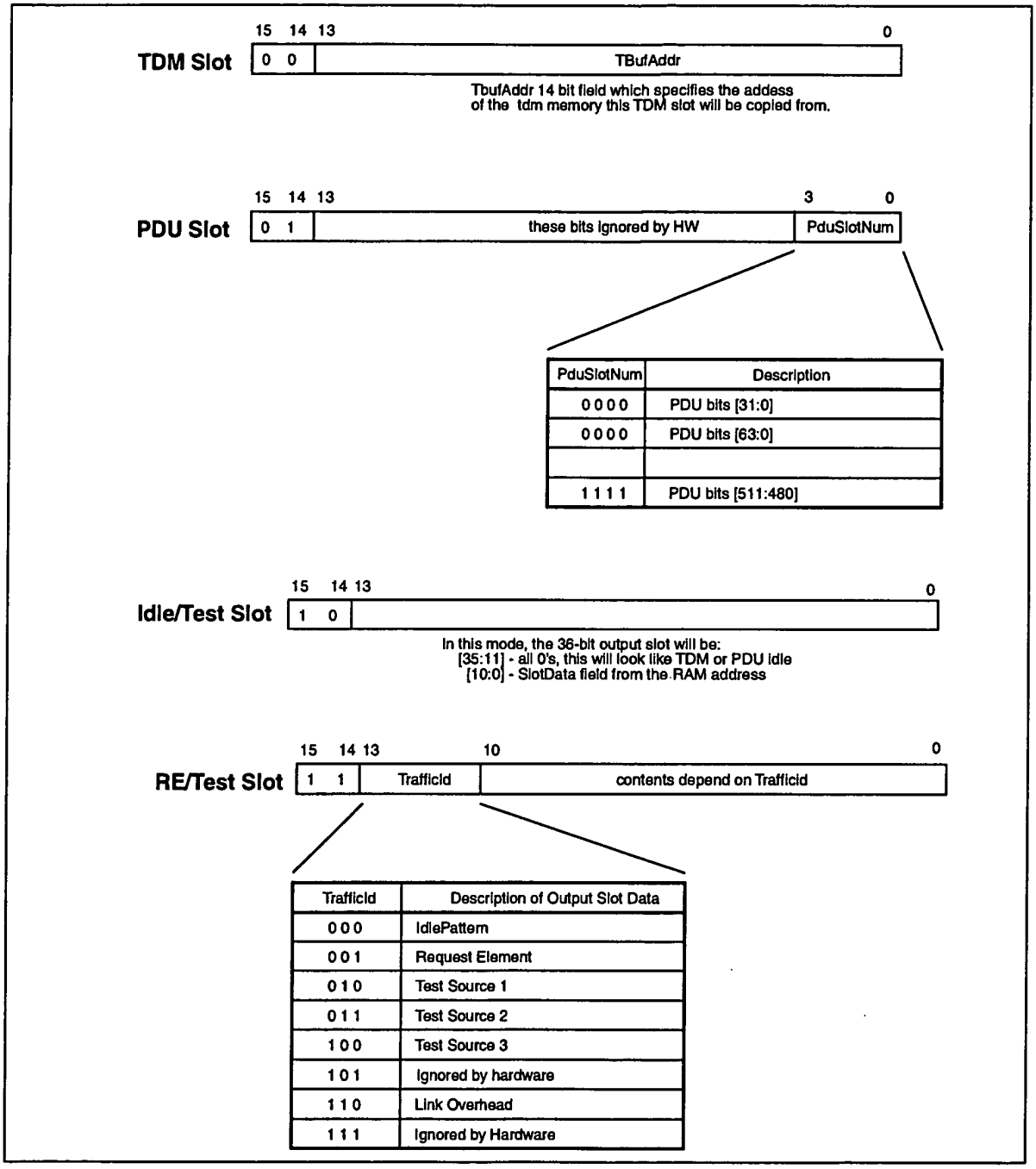


Figure 10-2: Mapper RAM Structure (1 of 2)

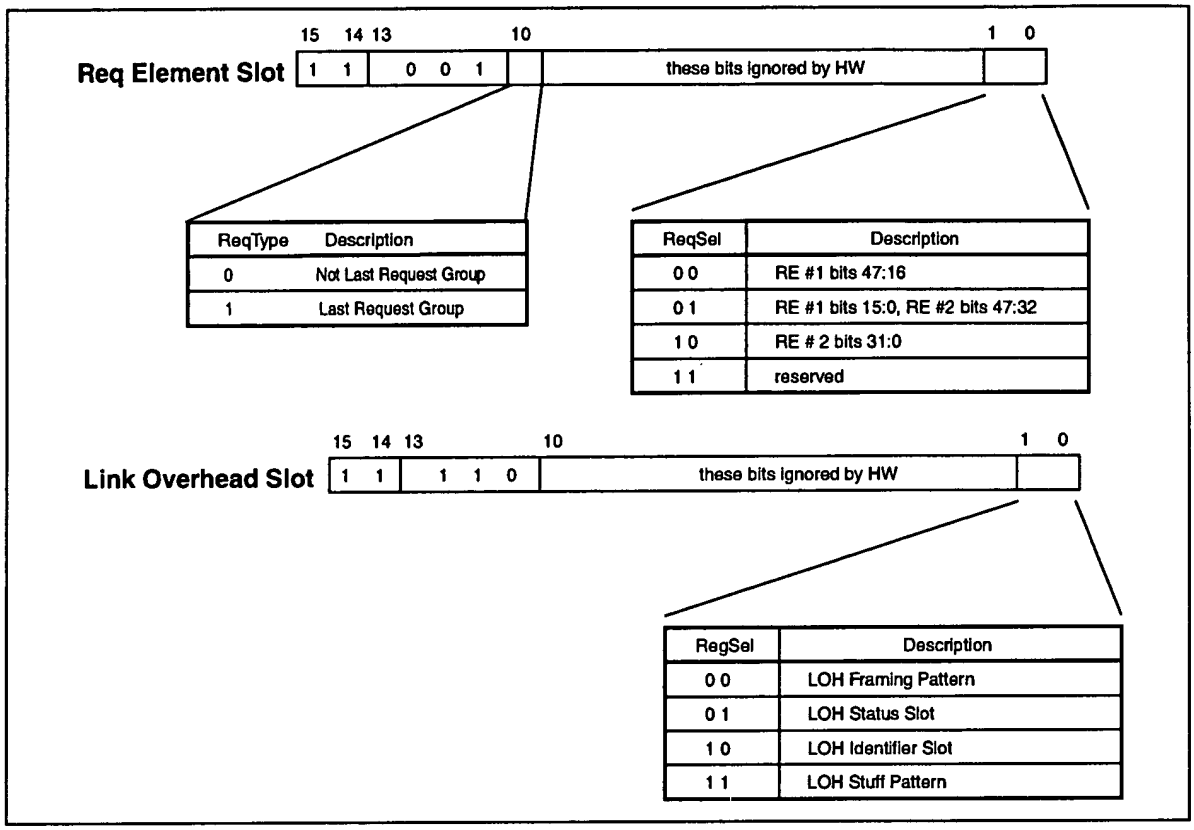


Figure 10-3: Mapper RAM Structure (2 of 2)

10.2 The ATM/IP Data Fifo

The ATM/IP Data Fifo is used as the interface between the Data Link Manager and the Data Mapper. A block diagram of the ATM/IP Data fifo is shown below in 10-4

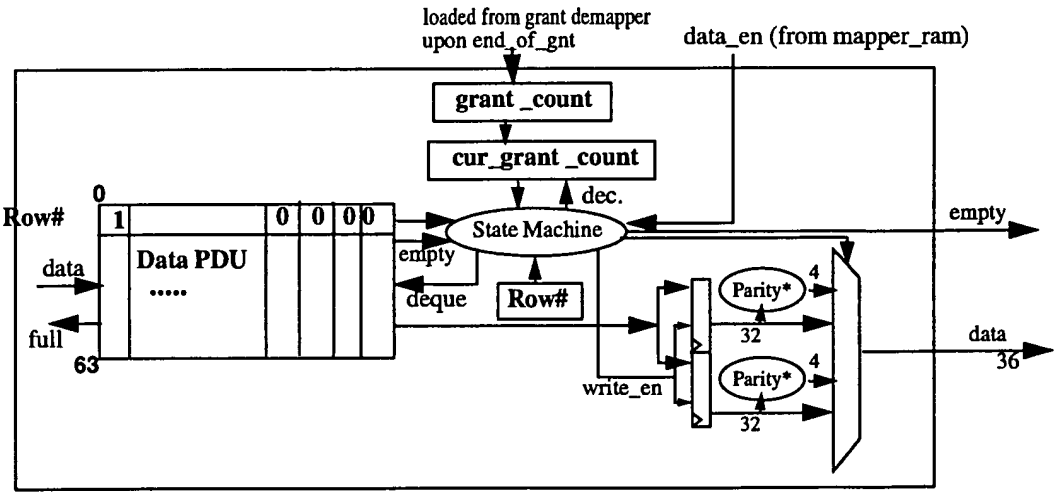


Figure 10-4: ATM/IP Data Fifo

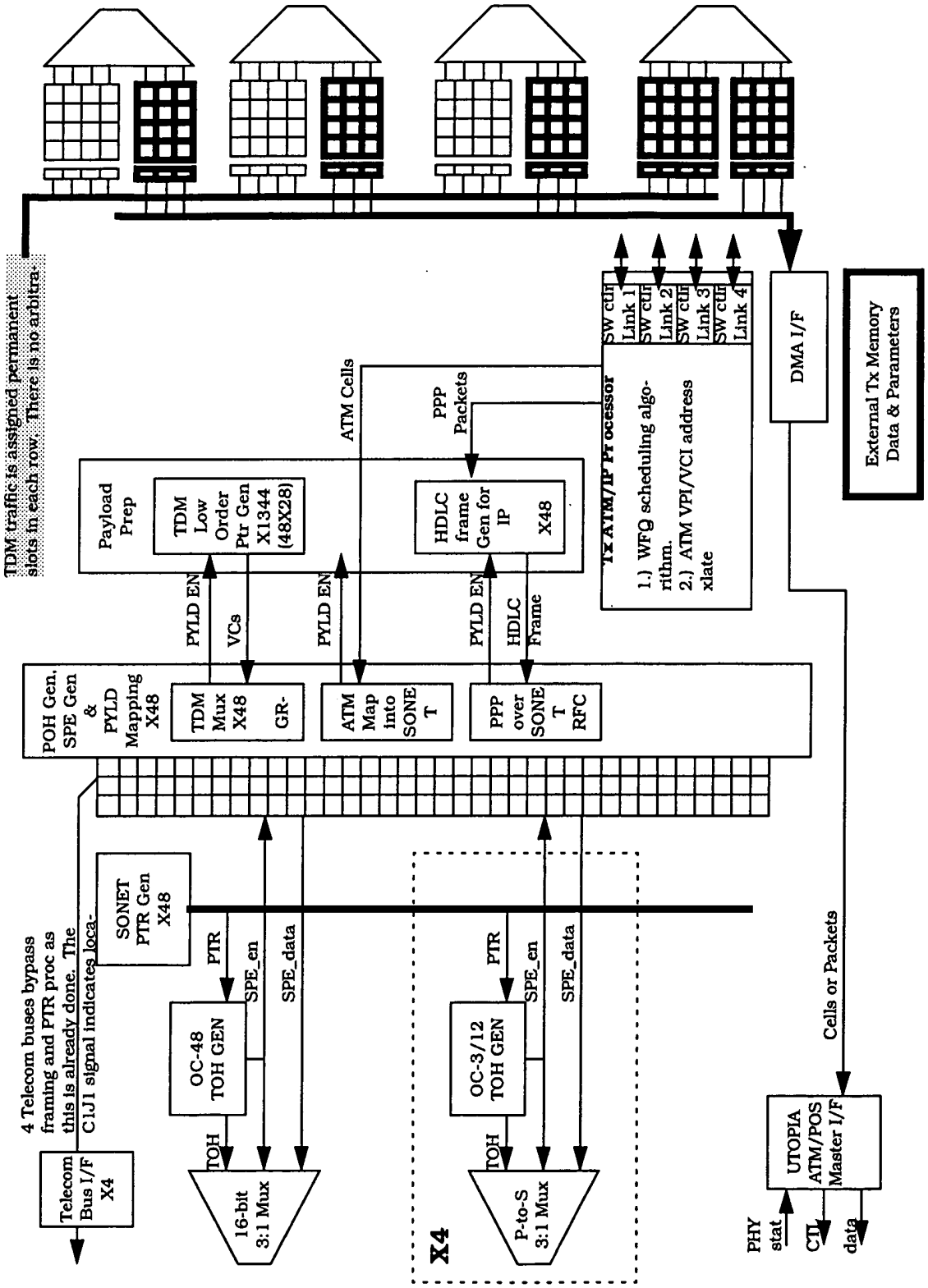
The Data Link Manager writes the Data PDU's using a 64 bit interface to the fifo. The data is transmitted from the fifo in 32 bit slots with 4 bits of parity. The State Machine associated with the ATM/IP data pdu fifo has to monitor the status of the fifo and maintain data integrity. The following checks are performed by the State Machine.

1. At the beginning of each row, i.e when sor_en_b is asserted, the state machine will check the cur_grant_count. If the cur_grant_count is > 0, an alarm will be raised. (If cur_grant_count > 0 this implies that there were data PDU's left over from the previous row in the fifo). The grant_count will be loaded into cur_grant_count. (The grant_count is loaded from the grant demapper at the assertion of the end_of_gnt signal)
2. The row# will be toggled. Upon startup it is set to the value opposite to the value of the row# in the arbiter block. This is to reflect the normal device operation, where a data slot granted in row i will be transmitted in the row i + 1.
3. The empty status line will be set if there are less than 7, double pdu data slots in the fifo. Upon receipt of a data_en from the mapper ram, the state_machine will check a 4 bit counter to see if it is zero (this counts the number of data_pdu slots available for transmission of a pdu). If zero and the empty status line is not asserted i.e not zero. (i.e there is at least one full data pdu available for transmission). The dequeue will be asserted and the write_en will be asserted along with a select bit to select the top half of the data pdu for transmission through the mux.
4. The count will be incremented upon each assertion of data_en and will reset to zero, if data_en is not asserted. The rationale for this counting behavior. The data mapper will have 16 consecutive 1's for loading a data pdu, and can be begin asserting immediately after transmitting a data-pdu, in which case the counter just rolled over to zero, or immediately after transmitting either TDM data or a request, in which case the counter will be reset to zero. The ATM/IP Fifo cannot begin transmitting a data pdu if any data_en have been asserted, as all 16 slots are required to transmit a data pdu and the header information for the pdu is found in the first few dataslots.???? (check with mike how he plans to do this since he is talking about less than 16 consecutive data PDU's icky!!!)
5. Upon transmission of a data pdu, the row# placed in the fifo will be checked against the row# in the state machine, if they match the data will be transmitted, else the data will be suppressed. A mismatch may happen if data-pdu's are left over from transmission of the previous row. This may happen if the data-link manager, was late in writing in data-pdus and the number of data-pdu slots left in the row to transmit the remaining data-pdus was less than the data-remaining in the fifo. The cur_grant_count will not be decremented. An alarm will be raised to inform the control processor of this error condition.
6. If the data-pdus row# matches the row# and cur_grant_count is greater than zero, then the data-pdu will be transmitted. (Conditions: empty not asserted and cur_grant_count > 0 are prerequisites too).
7. If the cur_grant_count is zero, and the data-pdu is not empty and data_en is asserted, no data will be transmitted. If the row# matches the row# in the state_machine, the data will be dequeued from the fifo and an alarm will be raised. This is because we cannot transmit more data-pdus than that have been granted. If the row# does not match the row# in the state_machine, the data PDU's will remain in the fifo, till the assertion of the sor_en_b at which time it can be transmitted in the next row.

10.3 Transmit Data Path

TDM, ATM and IP data are received through the switch interface ports. The data is routed to the correct processing blocks and then mapped into the line side interfaces. The TDM data is mapped directly into pre-assigned data slots. The ATM and IP data are routed to a Tx ATM/IP processor. This processor schedules these cells and packets for transmission using a WFQ algorithm and a leaky bucket rate shaper. The Tx processor also performs IP routing and VPI/VCI address translation.

10.3.1 Transmit Side Block Diagram



11 Transmit AIT/IP Traffic Processor

RESERVED

11.1 SFQ Scheduling Algorithm

RESERVED.

11.2 Start Number Sorting Algorithm

RESERVED

11.3 Hardware Heap Sorter

RESERVED

11.4 ATM Cell Processing

RESERVED.

11.4.1 ATM Cell Shaping

RESERVED.

11.4.2 ATM Cell Control Parameter Table

RESERVED

11.5 IP Packet Processing

RESERVED.

11.5.1 IP Packet Control Parameter Table

RESERVED

11.6 MPLS Packet Processing

RESERVED

11.6.1 MPLS Packet Control Parameter Table

RESERVED

11.7 Frame Relay Packet Processing

RESERVED

11.7.1 Frame Relay Packet Control Parameter Table

RESERVED.

11.8 Tx AIT Processor FIFO Data Structures

RESERVED

11.8.1 Shaping/Scheduling Parameter Read FIFO

RESERVED.

11.8.2 Shaping/Scheduling Parameter Write FIFO

RESERVED

11.8.3 External Memory Access Control FIFO

RESERVED

11.8.4 Control Message FIFO Format

RESERVED.

11.8.5 Enqueue Req FIFO

RESERVED

11.8.6 Tx AIT Dequeue Ack FIFO

RESERVED

11.8.7 Tx DLM Dequeue Req FIFO

RESERVED.

11.8.8 Tx DLM Dequeue Ack FIFO

RESERVED

11.8.9 FIFO and Hardware Heap Sorter Status Register

RESERVED.

11.9 Memory Map

RESERVED

12 Transmit Data Link Manager

The Transmit Data Link Manager (Tx DLM) manages the external tx memory which is used to store tx data and tx control parameters. All of the ATM cells and IP packets received from the switch interface are stored in the external tx memory while they wait to be scheduled and transmitted to an output SONET or UTOPIA port. Parameters required by the transmit shaping and scheduling algorithms are stored in the external tx memory. The Tx DLM will also perform address translation on ATM cells.

The Tx DLM will also process MPLS encapsulated IP packets. The scheduling function for MPLS will be the same as for IP. The main difference between IP and MPLS is that the Tx DLM must insert or delete an MPLS label when the Service Processor is at the edge of an MPLS network and must perform label switching when the Service Processor is in the core of an MPLS network.

The Tx DLM also allows the Host interface access to the external memory for writing and reading control parameters and to insert diagnostic and control ATM cells and IP and MPLS packets in the transmit path.

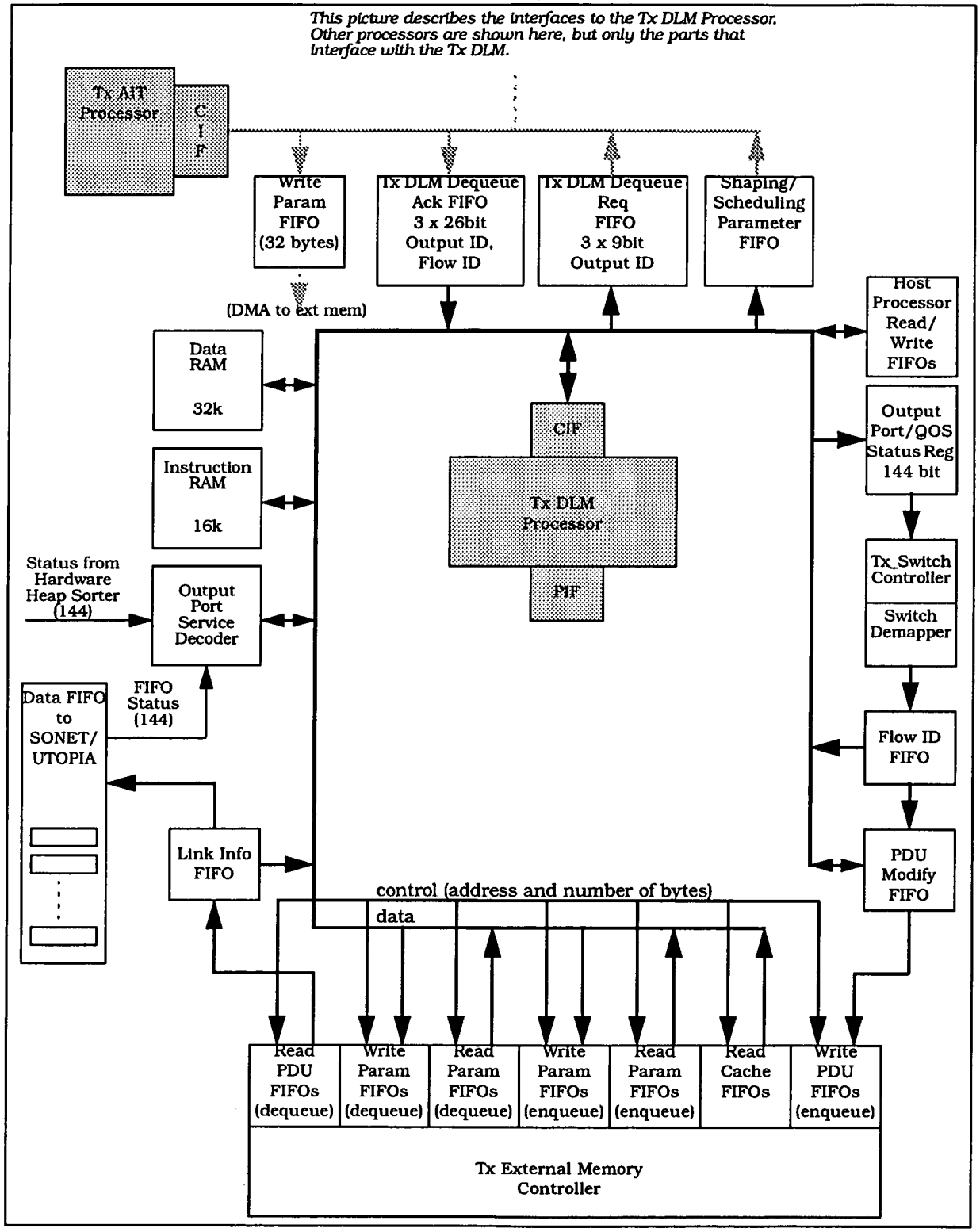


Figure 12-1: Tx DLM Block Diagram

The TxDLM is responsible for the tasks described in the following sections.

12.1 Tx Data Flow

The TxDLM stores ATM cells and IP and MPLS packet chunks in external memory.

- Maintain a linked list memory structure on a per-flow basis
- Maintain the Head, Tail and Length in units of 52-byte cells on a per-flow basis

The length is stored on a per-flow basis for diagnostic purposes. If a particular flow or set of flows is filling the buffer to an unusually high level, then something is wrong. Maybe the SONET or UTOPIA output port is not configured correctly. Whatever the reason the DLM will alert SW so that some corrective action can be taken.

The data flow from the Switch Demapper to external memory is as follows:

- The Switch Demapper writes a PDU to the PDU FIFO and interrupts the Tx DLM.
- The Tx DLM reads the header information from the PDU FIFO, and extracts the Flow ID.
- Based on the Flow ID, the Tx DLM retrieves the Linked List/Shaping/Scheduling data structure from external memory.
- The Tx DLM writes the linked list pointer to the PDU FIFO and performs ATM address translation or MPLS label swapping, addition or deletion. The Tx DLM then initiates a DMA transfer to move the PDU to external memory.
- The Tx DLM updates the tail and count fields in the Linked List/Shaping/Scheduling data structure.
- If the PDU is an ATM cell or the last PDU of an IP or MPLS packet, then the Tx DLM passes the data structure to the Shaping/Scheduling processor through the Shaping/Scheduling Parameter FIFO. If the PDU is the first or middle fragments of an IP packet, the Tx DLM will write the data structure directly back to external memory.

The Tx AIT processor will performing the Shaping and Scheduling functions and then update and write the Linked List/Shaping/Scheduling data structure back to external memory. The data flow from external memory to the SONET/UTOPIA Data FIFOs is as follows:

- The Tx DLM will poll the Output Port Service Decoder to determine which SONET or UTOPIA Data FIFO needs servicing. The Output Port Service Decoder will perform a round robin poll of the SONET and UTOPIA Data FIFO not full signals and the Hardware Heap Sorter status signals. If a packet has been scheduled and the Data FIFO is not full, then the Output ID will be presented by the Output Port Service Decoder.
- The Tx DLM will write the Output ID to the Tx DLM Dequeue Req FIFO. The Tx AIT processor will transfer the Output ID to the Hardware Heap Sorter module. When the Hardware Heap Sorter has retrieved the Flow ID from the heap, the Tx AIT processor will write the Flow ID and Output ID to the Tx DLM Dequeue Ack FIFO.
- For a multi-PDU IP or MPLS packet, the Tx DLM will write the Output ID to Tx DLM Dequeue Req FIFO only for the Start of Packet (SOP). After that the Tx DLM will set a mask bit in the Output Port Service Decoder so that only the SONET and UTOPIA Data FIFO not full flag is used by the decoder.
- When a Flow ID and Output ID has been retrieved from the Hardware Heap Sorter, the Tx AIT will write the values to the Tx DLM Dequeue Ack FIFO. Tx DLM will then initiate a DMA transfer from external memory to the SONET/UTOPIA FIFO for the Flow ID.
- The Tx DLM will update the Link List/Shaping/Scheduling data structure and write it back to external memory.

The linked list structure for each flow is illustrated in Figure 12-2 below:

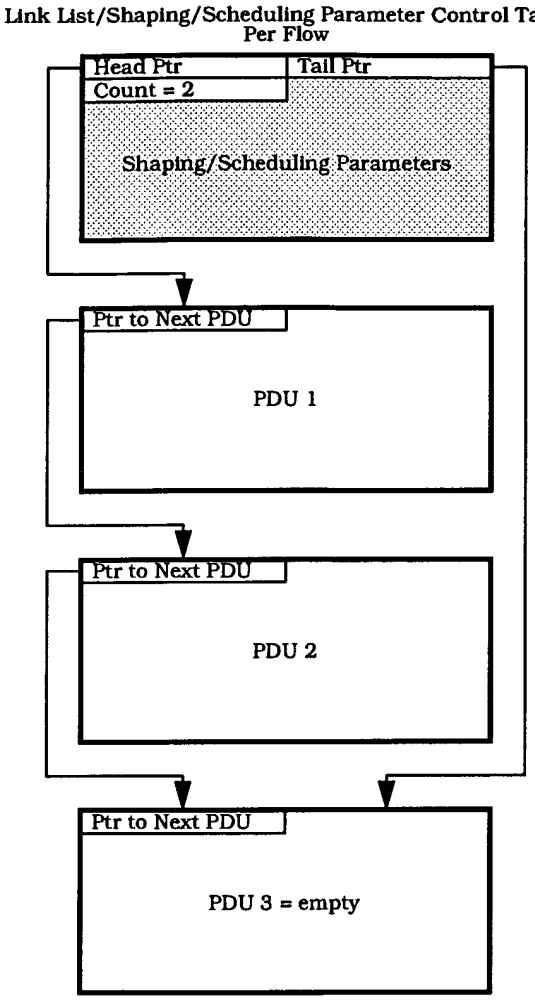
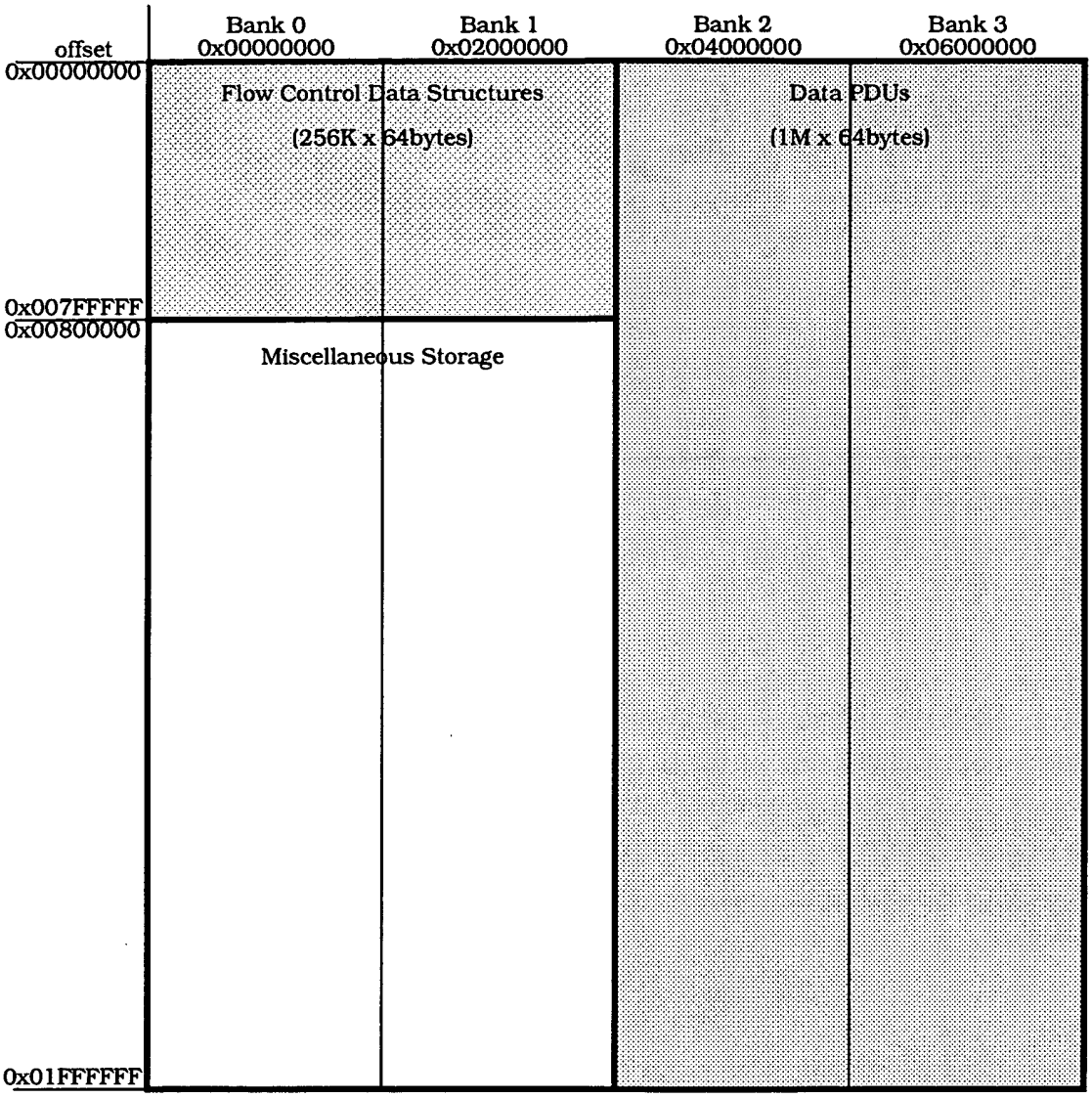


Figure 12-2: Linked List Illustration

12.2 External Memory Map

The external memory will consists of a quantity of 4, "4-bank X 4-Meg X 16-bit" FCRAM memory chips, for a total of 132M bytes of memory. The memory will partitioned such that the control table data structures will be stored in banks 0 and 1, and the PDU data structures will be stored in

banks 2 and 3.



12.3 PDU Data Formats

The format of the PDU received from the switch is shown in Table Figure 12-3 below.

Table 12-3: PDU Format from Switch

6 3	5 5 6 5	4 4 8 7	4 3 0 9	3 3 2 1	2 2 4 3	1 1 6 5	8 7	0
Pdu	-	-	RouteTag		Ver	ValidPlBytes	VOQID	Frg	A	-	-	-	-	-	-	-	-	-	-	SeqNo
-	-	-	-	-	-	-	-	-	-	-	-	-	-	FlowID		Payload_Byte_0		Payload_Byte_1		Payload_Byte_2		Payload_Byte_3	
Payload_Byte_4		Payload_Byte_5		Payload_Byte_6		Payload_Byte_7		Payload_Byte_8		Payload_Byte_9		Payload_Byte_10		Payload_Byte_11	
Payload_Byte_12		Payload_Byte_13		Payload_Byte_14		Payload_Byte_15		Payload_Byte_16		Payload_Byte_17		Payload_Byte_18		Payload_Byte_19	
Payload_Byte_20		Payload_Byte_21		Payload_Byte_22		Payload_Byte_23		Payload_Byte_24		Payload_Byte_25		Payload_Byte_26		Payload_Byte_27	
Payload_Byte_28		Payload_Byte_29		Payload_Byte_30		Payload_Byte_31		Payload_Byte_32		Payload_Byte_33		Payload_Byte_34		Payload_Byte_35	
Payload_Byte_36		Payload_Byte_37		Payload_Byte_38		Payload_Byte_39		Payload_Byte_40		Payload_Byte_41		Payload_Byte_42		Payload_Byte_43	
Payload_Byte_44		Payload_Byte_45		Payload_Byte_46		Payload_Byte_47		Payload_Byte_48		Payload_Byte_49		Payload_Byte_50		Payload_Byte_51	

Note: reserved bit positions are indicated with a "-". The default state for these bits is 0.

The field descriptions are as follows:

Pdu

This 2-bit field identifies what type of data is being carried within this PDU.

PDU[1:0]	Description
00	Idle
01	ATM Cell
10	IP Packet
11	Control Message

If an "Idle" PDU is detected, it will not be forwarded through the switch.

RouteTag

28-bit field which identifies the self route through the switch fabric.

A - Abort

This bit will be set if fragmentation for this packet is being aborted. When the Tx DLM detects a PDU with this bit set, it will terminate reassembly of the packet and discard any fragments previously received.

Ver

This 2-bit field indicates the iTAP protocol version used when creating this PDU. This field will be set to "00" for the initial version of the iTAP chipset.

ValidPlBytes

For IP and control PDUs, when the Fragment Indicator is set for "Last Fragment" this 6-bit field will indicate how many payload bytes are carried in this PDU.

VOQID

This 5-bit field identifies the destination iTPP's Virtual Output Queue for this PDU.

Frg

This 2-bit field identifies whether this is a complete packet or a fragment of a longer IP packet or control message.

Frg[1:0]	Description
00	Middle Fragment
01	First Fragment

10	Last Fragment
11	Complete Packet

A - Abort

This bit will be set if fragmentation for this packet is being aborted. When the output port processor detects a PDU with this bit set, it will terminate reassembly of the packet and discard any fragments previously received.

SeqNo

SeqNo is the fragment sequence count, it will be incremented for each fragment. The SeqNo will start at 0 for the first fragment. If there are more than 16 fragments to the PDU, this SeqNo will roll over past 15 and continue counting.

FlowId

This 17-bit field identifies which flow this cell belongs to in the destination iTPP.

After the entire PDU has been received in the Enqueue PDU FIFO, the Tx DLM will modify the PDU by adding the Next Pointer for link list operation and by performing ATM address translation or MPLS label swapping, insertion, or deletion. The format of the modified PDU is shown in Table Figure 12-4.

Table 12-4: PDU Format in External Memory

6 3	5 5 6 5	4 4 8 7	4 3 0 9	3 3 2 1	2 2 4 3	1 1 6 5	8 7	0

PhyNo	Ver	ValidPlBytes	VOQID	Frg	A	-	-	PduStart	NextPointer			
(may contain data if an MPLS flow)									Payload_Byte_0	Payload_Byte_1	Payload_Byte_2	Payload_Byte_3
Payload_Byte_4	Payload_Byte_5	Payload_Byte_6	Payload_Byte_7						Payload_Byte_8	Payload_Byte_9	Payload_Byte_10	Payload_Byte_11
Payload_Byte_12	Payload_Byte_13	Payload_Byte_14	Payload_Byte_15						Payload_Byte_16	Payload_Byte_17	Payload_Byte_18	Payload_Byte_19
Payload_Byte_20	Payload_Byte_21	Payload_Byte_22	Payload_Byte_23						Payload_Byte_24	Payload_Byte_25	Payload_Byte_26	Payload_Byte_27
Payload_Byte_28	Payload_Byte_29	Payload_Byte_30	Payload_Byte_31						Payload_Byte_32	Payload_Byte_33	Payload_Byte_34	Payload_Byte_35
Payload_Byte_36	Payload_Byte_37	Payload_Byte_38	Payload_Byte_39						Payload_Byte_40	Payload_Byte_41	Payload_Byte_42	Payload_Byte_43
Payload_Byte_44	Payload_Byte_45	Payload_Byte_46	Payload_Byte_47						Payload_Byte_48	Payload_Byte_49	Payload_Byte_50	Payload_Byte_51

The PDU fields modified by the Tx DLM are described below:

PhyNo

Phy No is a value from decimal 0 to 143, that identifies which SONET or UTOPIA Data FIFO the PDU is to be written to when the PDU is dequeued from external memory.

NextPointer

This 32-bit field is used to link the PDUs for a particular flow together.

PduStart

This 6-bit field will be used during the dequeue operation to determine the location of the first payload byte. The default location is 12, as shown in Table Figure 12-4. In applications where an MPLS label is being inserted, the first payload byte will be in location 8. Likewise, in applications where an MPLS header is being deleted, the first payload byte will be in location 16.

12.4 ATM Cell Processing

The data flow for receiving an ATM cell PDU from the switch, performing address translation and then enqueueing the cell PDU in external memory is described below:

- The Switch Demapper writes an ATM cell to the Enqueue PDU FIFO.

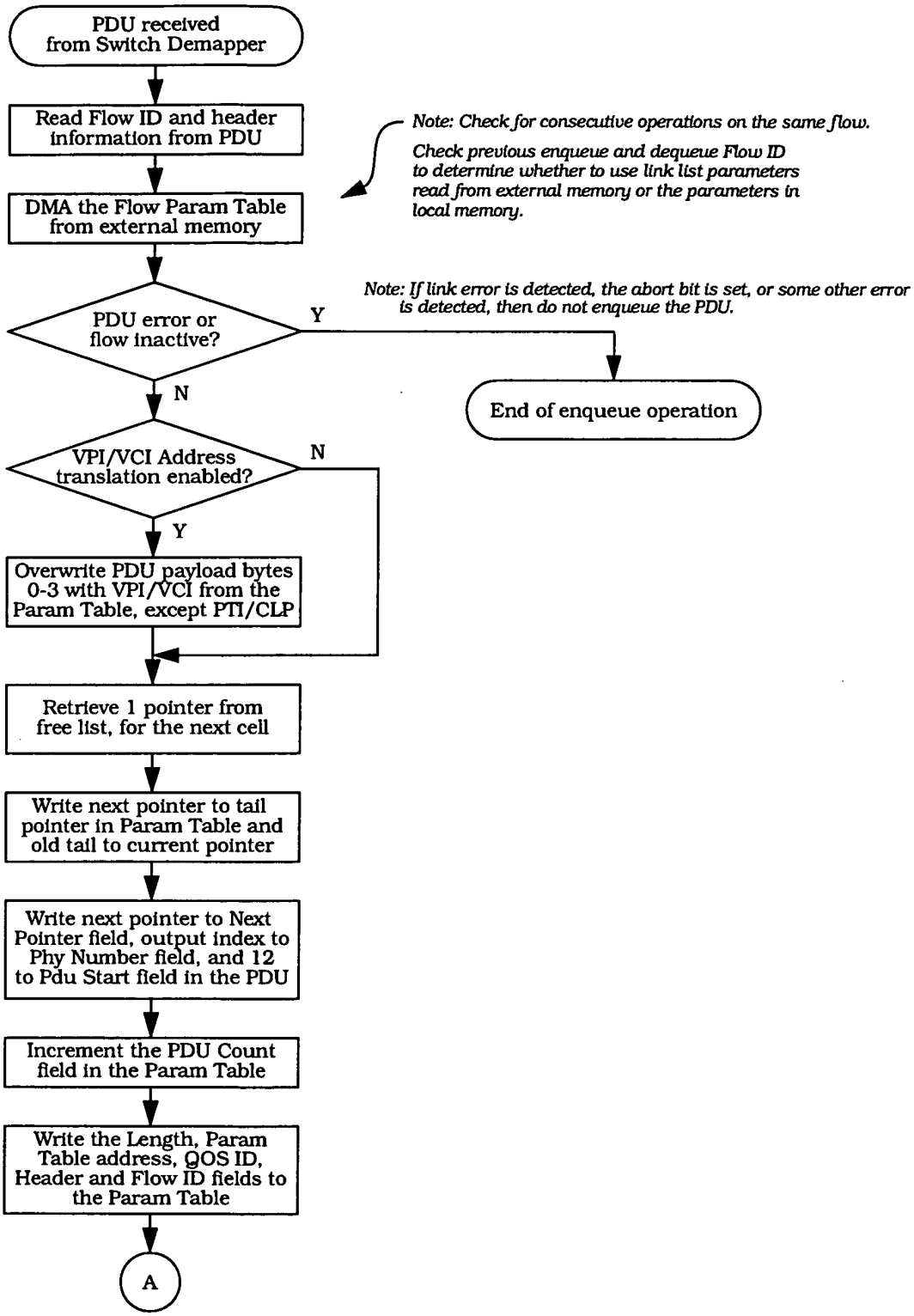
- The Tx DLM will begin processing the ATM cell when the Enqueue PDU FIFO full flag is set. The full flag can either be polled or an interrupt can be generated. Hardware supports either method.
- The Tx DLM will read the Flow ID from the FIFO and retrieve the Linked List/Shaping/ Scheduling parameter table from external memory.
- A new PDU pointer will be allocated from the Free List and written to the NextPointer field.
- The address translation enable bit in the parameter table will be examined to determine if the VPI/VCI address bytes in the PDU should be overwritten with values from the parameter table. If enabled, PDU bytes 0-3 will be overwritten with the VPI/VCI from the parameter table, except for the PTI and CLP bits.
- The parameter table is transferred to the Shaping/Scheduling Parameter FIFO so that the ATM cell can be scheduled.

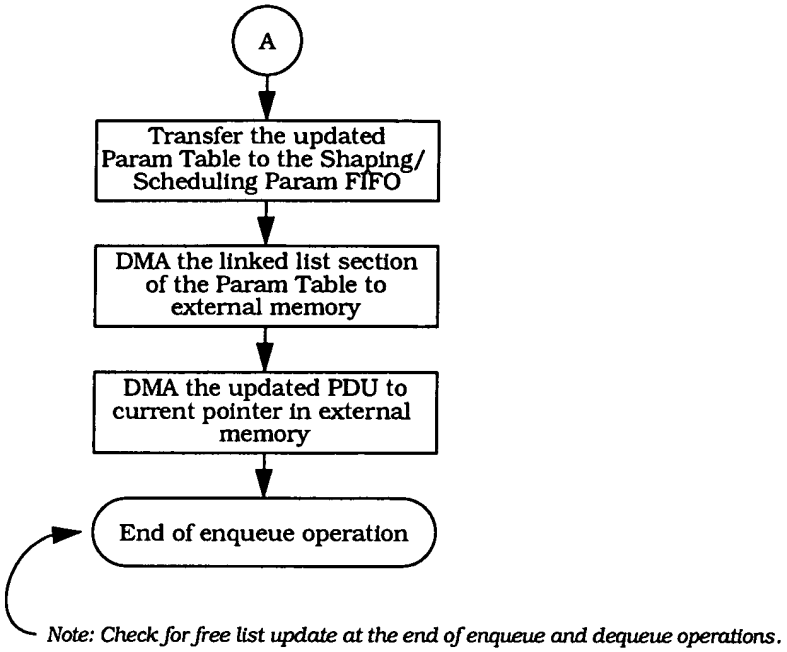
The address translation function and resulting format of the ATM cell PDU in external memory is shown in the table below.

Table 12-5: ATM Cell PDU Format to External Memory

6 3	5 5 6 5	4 4 8 7	4 3 0 9	3 3 2 1	2 2 4 3	1 1 6 5	8 7	0
PhyNo	Ver	ValidPlBytes	VOQID	Frg	A	-	-	PduStart	NextPointer			
- - - - - - - -	- - - - - - - -	- - - - - - - -	- - - - - - - -	- -	-	- - - - - - - -	VPI/VCI Byte 0	VPI/VCI Byte 1	VPI/VCI Byte 2	VPI/VCI Byte 3
Payload_Byte_4	Payload_Byte_5	Payload_Byte_6	Payload_Byte_7	Payload_Byte_8	Payload_Byte_9	Payload_Byte_10	Payload_Byte_11			
Payload_Byte_12	Payload_Byte_13	Payload_Byte_14	Payload_Byte_15	Payload_Byte_16	Payload_Byte_17	Payload_Byte_18	Payload_Byte_19			
Payload_Byte_20	Payload_Byte_21	Payload_Byte_22	Payload_Byte_23	Payload_Byte_24	Payload_Byte_25	Payload_Byte_26	Payload_Byte_27			
Payload_Byte_28	Payload_Byte_29	Payload_Byte_30	Payload_Byte_31	Payload_Byte_32	Payload_Byte_33	Payload_Byte_34	Payload_Byte_35			
Payload_Byte_36	Payload_Byte_37	Payload_Byte_38	Payload_Byte_39	Payload_Byte_40	Payload_Byte_41	Payload_Byte_42	Payload_Byte_43			
Payload_Byte_44	Payload_Byte_45	Payload_Byte_46	Payload_Byte_47	Payload_Byte_48	Payload_Byte_49	Payload_Byte_50	Payload_Byte_51			

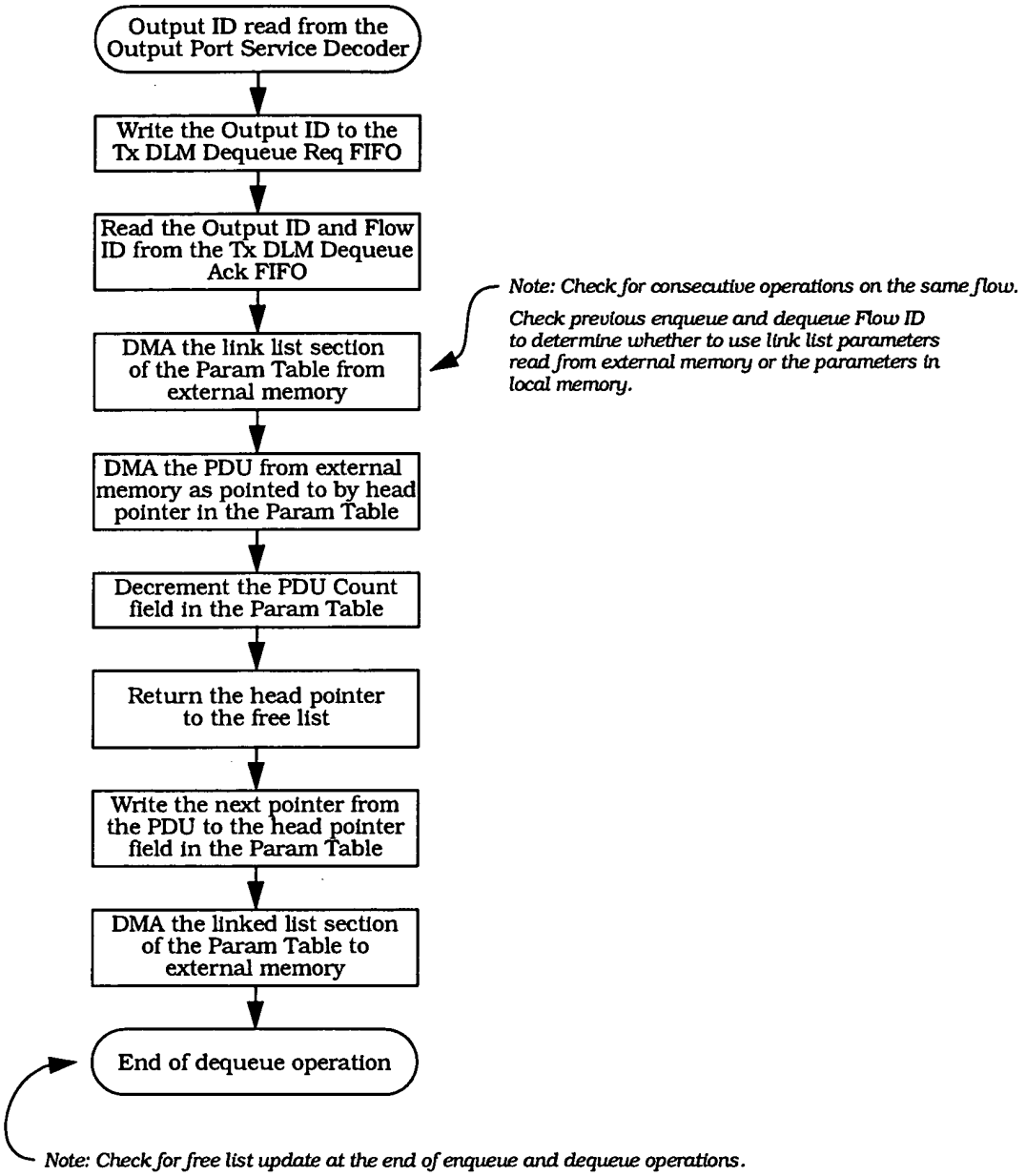
The enqueue process for an ATM cell PDU is shown in the flow chart below:





Note: Check for free list update at the end of enqueue and dequeue operations.

The dequeue process for an ATM cell PDU is shown in the flow chart below:



The shaping, scheduling, and state information that must be kept for each flow is described in this section. The table format for an ATM cell flow is shown in the table below.

Table 12-6: ATM Flow Transmit Control Table

63	55	44	43	33	22	11	87	0	addr offset
Last Finishing Number (dynamic)					Egress CLP0+1 Counter (dynamic)				0x00
Last Conformance Timestamp 0 (dynamic)					Last Conformance Timestamp 1 (dynamic)				0x08
Shaping Bucket 0 Counter (dynamic)					Shaping Bucket 1 Counter (dynamic)				0x10
Non-Conforming Weight, Bucket 0 (static)		Non-Conforming Weight, Bucket 1 (static)			Non-Conforming Cell Counter (dynamic)		OAM Cell Counter (dynamic)		0x18
Shaping Bucket 0 Limit (static)		Shaping Bucket 1 Limit (static)			LmtMult0 (static)	LmtMult1 (static)	Conforming Weigh (static)		0x20
Shaping Bucket 0 Increment (static)		Shaping Bucket 1 Increment (static)			VPI/VCI Address Translation (static)				0x28
MiscParams (static)	OutputIndex (static)	Shp0 (static)	Shp1 (static)		Head Pointer (dynamic)				0x30
PDU Count (dynamic)					Tail Pointer (dynamic)				0x38

12.5 IP Packet Processing

The data flow for receiving an IP PDU from the switch and enqueueing the IP PDU in external memory is described below:

- The Switch Demapper writes an IP PDU to the Enqueue PDU FIFO.
- The Tx DLM will begin processing the IP PDU when the Enqueue PDU FIFO full flag is set. The full flag can either be polled or an interrupt can be generated. Hardware supports either method.
- The Tx DLM will read the Flow ID from the FIFO and retrieve the Linked List/Shaping/ Scheduling parameter table from external memory.
- A new PDU pointer will be allocated from the Free List and written to the NextPointer field.
- If the PDU is the last DPU fragment of an IP packet or a single PDU IP packet (Frg field = 10 or 11), then the Tx DLM will transfer the parameter table to the Shaping/Scheduling Parameter FIFO so that the IP packet can be scheduled.

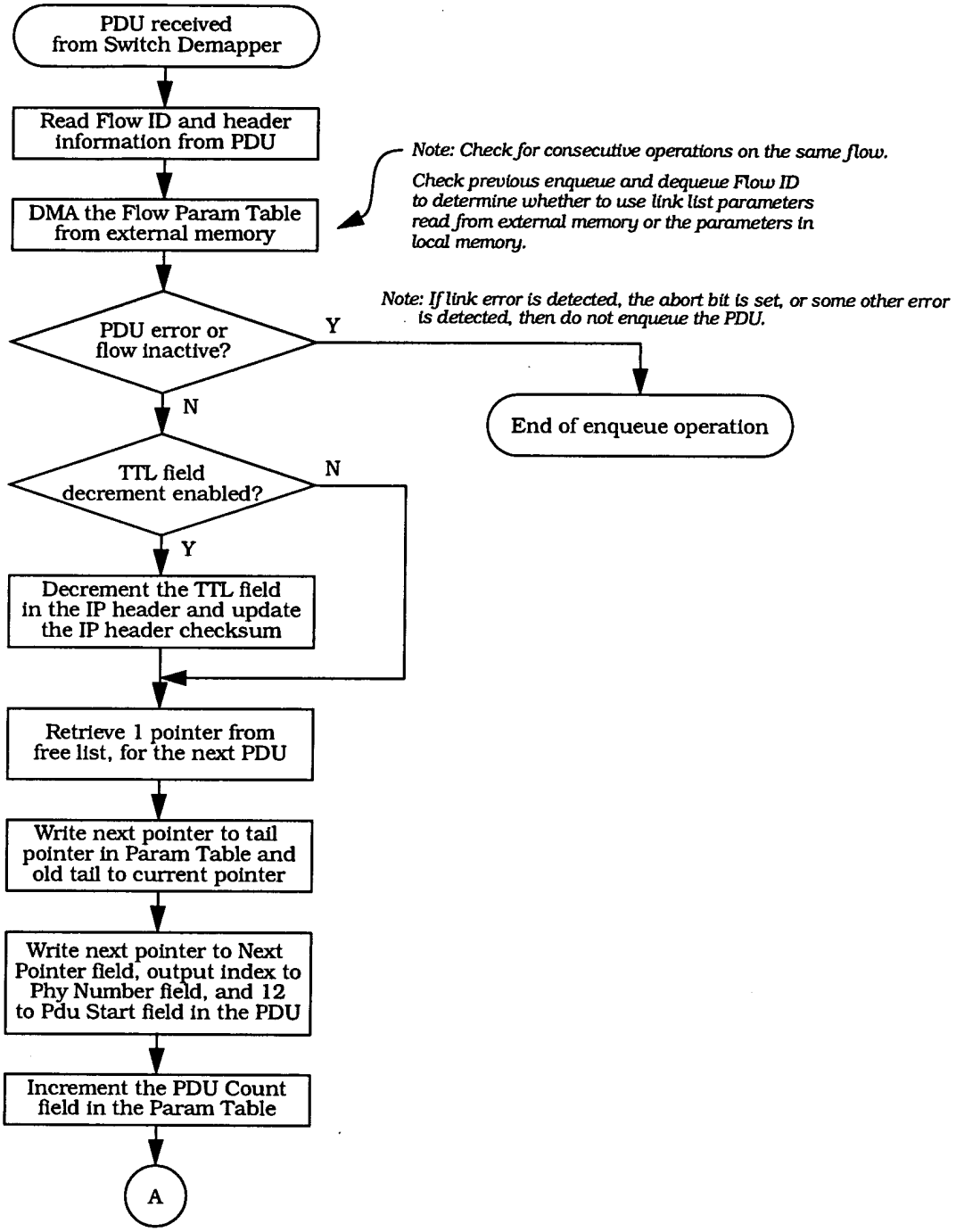
The format of the IP packet PDU in external memory is shown in the table below.

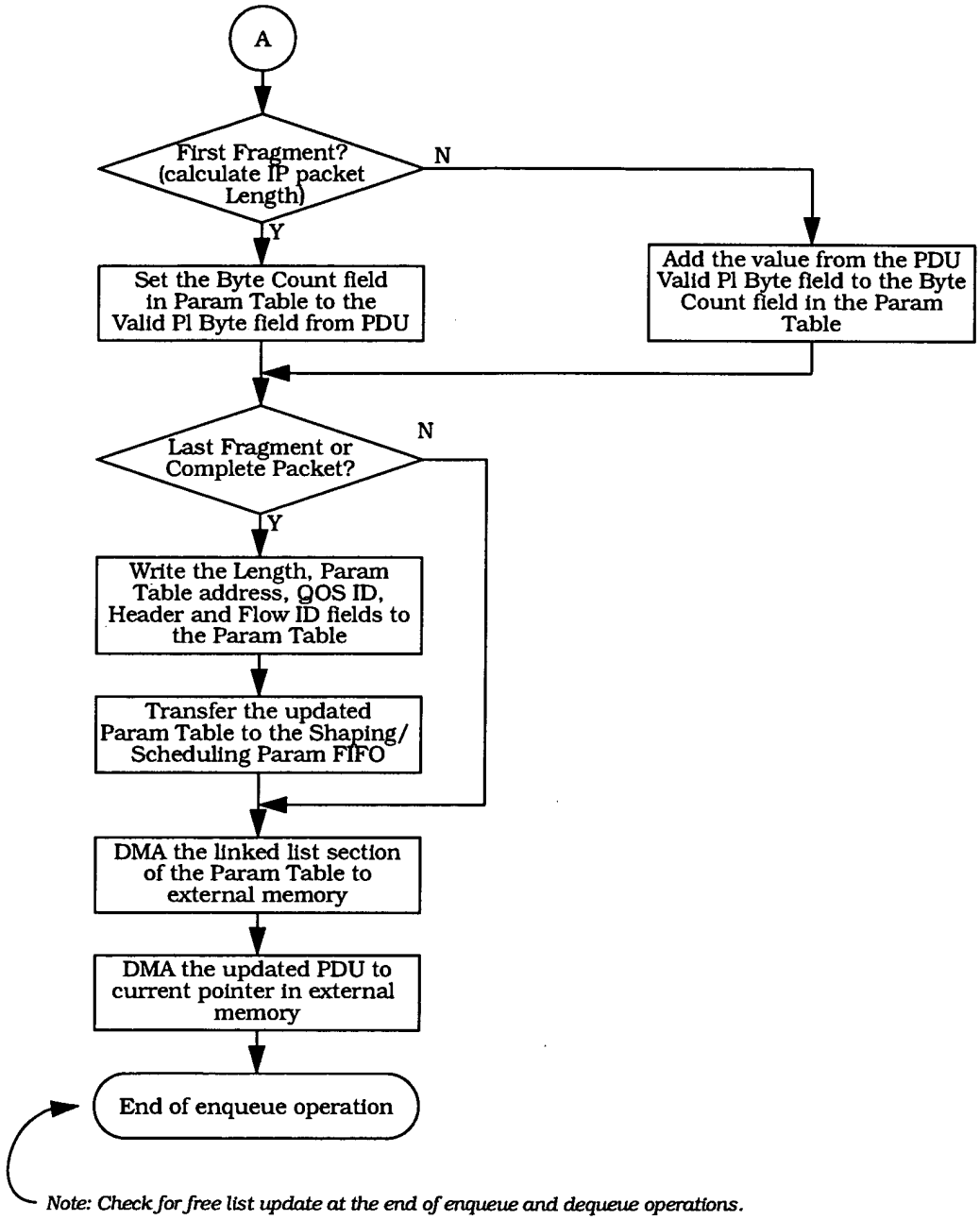
Table 12-7: IP Packet PDU Format to External Memory

63	55	44	43	33	22	11	87	0
PhyNo	Ver	ValidPlBytes	VOQID	Frg	A	-	-	PduStart	NextPointer			
- - - - - - - -	- - - - - - - -	- - - - - - - -	- - - - - - - -	- -	-	-	-	- - - - - - - -	Payload_Byte_0	Payload_Byte_1	Payload_Byte_2	Payload_Byte_3
Payload_Byte_4	Payload_Byte_5	Payload_Byte_6	Payload_Byte_7						Payload_Byte_8	Payload_Byte_9	Payload_Byte_10	Payload_Byte_11
Payload_Byte_12	Payload_Byte_13	Payload_Byte_14	Payload_Byte_15						Payload_Byte_16	Payload_Byte_17	Payload_Byte_18	Payload_Byte_19
Payload_Byte_20	Payload_Byte_21	Payload_Byte_22	Payload_Byte_23						Payload_Byte_24	Payload_Byte_25	Payload_Byte_26	Payload_Byte_27
Payload_Byte_28	Payload_Byte_29	Payload_Byte_30	Payload_Byte_31						Payload_Byte_32	Payload_Byte_33	Payload_Byte_34	Payload_Byte_35

6 3	5 5 6 5	4 4 8 7	4 3 0 9	3 3 2 1	2 2 4 3	1 1 6 5	8 7	0
Payload_Byte_36	Payload_Byte_37	Payload_Byte_38	Payload_Byte_39	Payload_Byte_40	Payload_Byte_41	Payload_Byte_42	Payload_Byte_43	
Payload_Byte_44	Payload_Byte_45	Payload_Byte_46	Payload_Byte_47	Payload_Byte_48	Payload_Byte_49	Payload_Byte_50	Payload_Byte_51	

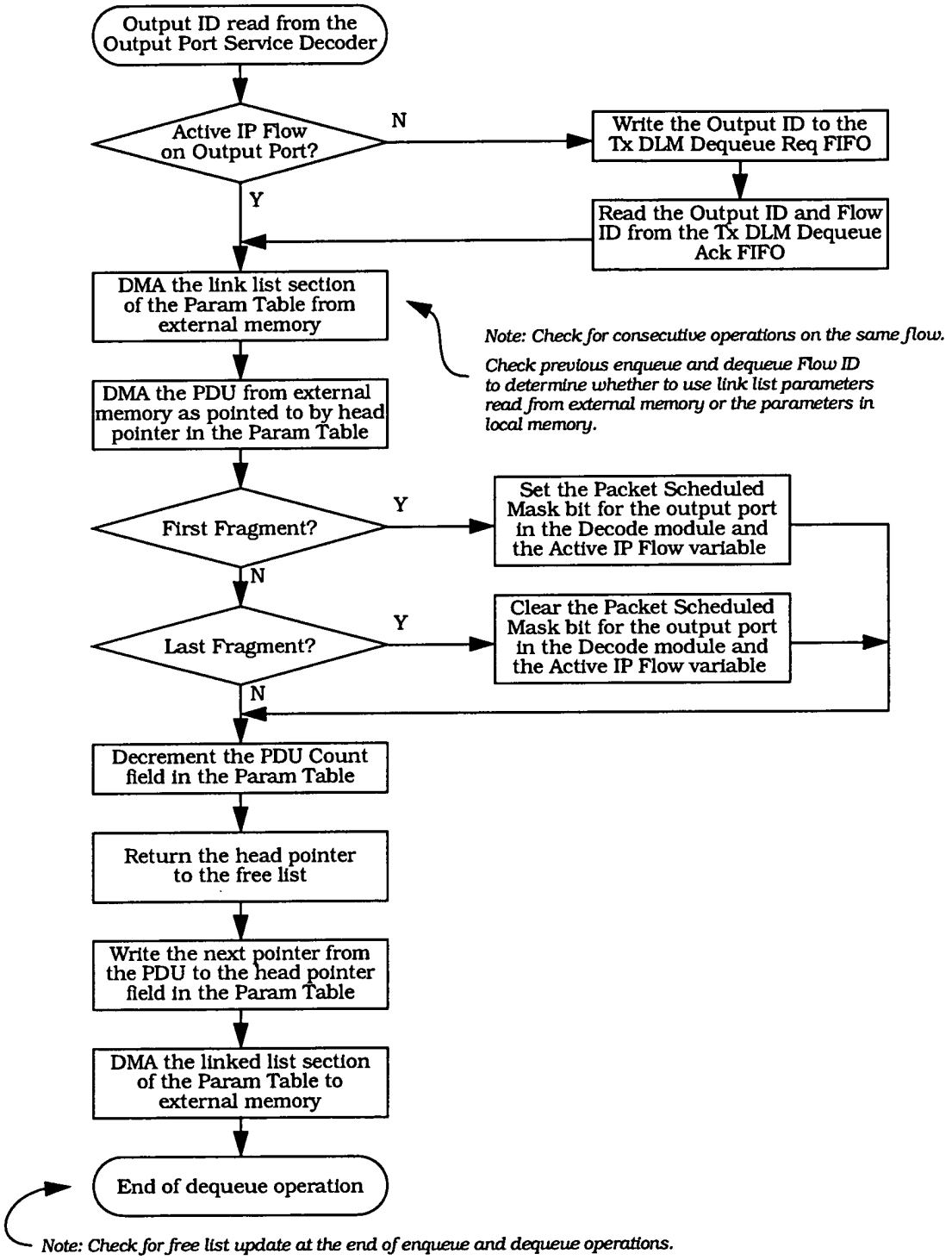
The enqueue process for an IP packet PDU is shown in the flow chart below:





Note: Check for free list update at the end of enqueue and dequeue operations.

The dequeue process for an IP packet PDU is shown in the flow chart below:



The control table data structure for an IP flow is shown in the table below:

Table 12-8: IP Flow Transmit Control Table

63	55	44	43	33	22	11	8 7	0	addr offset
Last Finishing Number (dynamic)				IP Byte Count, Cumulative (dynamic)					0x00
Reserved				Reserved					0x08
Reserved				Reserved					0x10
Reserved				Reserved					0x18
Enqueue Head Pointer (dynamic)				PDU Count in Current Packet (dynamic)		Flow Allocation Weight (static)			0x20
Dequeue Head Pointer (dynamic)				Reserved					0x28
MiscParams (static)	OutputIndex (static)	Byte Count in Current Packet (dynamic)		Head Pointer (dynamic)					0x30
PDU Count (dynamic)				Tail Pointer (dynamic)					0x38

12.6 MPLS Packet Processing

The data flow for receiving an MPLS PDU from the switch and enqueueing the MPLS PDU in external memory is described below:

- The Switch Demapper writes an MPLS PDU to the Enqueue PDU FIFO.
- The Tx DLM will begin processing the MPLS PDU when the Enqueue PDU FIFO full flag is set. The full flag can either be polled or an interrupt can be generated. Hardware supports either method.
- The Tx DLM will read the Flow ID from the FIFO and retrieve the Linked List/Shaping/ Scheduling parameter table from external memory.
- A new PDU pointer will be allocated from the Free List and written to the NextPointer field.
- The MplsMode and PppMode bits in the parameter table will be examined to determine what action should be taken with regards to the MPLS header. This is described in detail later in this section.
- If the PDU is the last DPU fragment of the MPLS packet or a single PDU MPLS packet (Frg field = 10 or 11), then the Tx DLM will transfer the parameter table to the Shaping/Scheduling Parameter FIFO so that the MPLS packet can be scheduled.

There are two MPLS operating modes that the Tx DLM must support: as a Label Edge Router (LER) and as a Label Switching Router (LSR). When functioning as an LER, the Tx DLM will be able to add and delete an MPLS label from an IP packet. When functioning as an LSR in the core of an MPLS network, the Tx DLM will perform MPLS label switching, similar to ATM address translation. An added complexity is that the IP packet may have a PPP protocol field appended to the front of the packet. The exact mode of operation is determined by the MplsMode field in the Link List/Shaping/ Scheduling parameter table. The MplsMode field is described below:

MplsMode[1:0]	Description
00	MPLS label added
01	MPLS label deleted
10	MPLS label switched
11	MPLS label unchanged

The location of the MPLS label in the PDU is determined by the PppMode bit, as described below:

PppMode	Description
0	IP packet, no PPP encapsulation
1	IP packet with PPP encapsulation

Several PDU examples are shown below, illustrating how the PDU format changes depending

on the MplsMode and PppMode bit settings:

PHY	PlBytes		PduStart				
				B0	B1	B2	B3
B4	B5	B6	B7	B8	B9	B10	B11
B12	....	.....	.....	.....	...		B19
B20	....	.....	.....	.....	...		B27
B28	....	.....	.....	.....	...		B35
B36	....	.....	.....	.....	...		B43
B44	B45	B46	B47	B48	B49	B50	B51

MPLS SOP PDU before label modification or MplsMode = 11
PduStart = 12

Legend: B - Payload Bytes
M - New MPLS Label Bytes

PHY	PlBytes		PduStart				
M0	M1	M2	M3	B0	B1	B2	B3
B4	B5	B6	B7	B8	B9	B10	B11
B12	....	.....	.....	.....	...		B19
B20	....	.....	.....	.....	...		B27
B28	....	.....	.....	.....	...		B35
B36	....	.....	.....	.....	...		B43
B44	B45	B46	B47	B48	B49	B50	B51

Add MLPS Label, no PPP encapuslation
MPLS SOP PDU after label modification
MplsMode = 00, PppMode = 0, PduStart = 8
PlBytes = ValidPlBytes + 4

PHY	PlBytes		PduStart				
B0	B1	M0	M1	M2	M3	B2	B3
B4	B5	B6	B7	B8	B9	B10	B11
B12	....	.....	.....	.....	...		B19
B20	....	.....	.....	.....	...		B27
B28	....	.....	.....	.....	...		B35
B36	....	.....	.....	.....	...		B43
B44	B45	B46	B47	B48	B49	B50	B51

Add MLPS Label, with PPP encapuslation
MPLS SOP PDU after label modification
MplsMode = 00, PppMode = 1, PduStart = 8
PlBytes = ValidPlBytes + 4

PHY	PlBytes		PduStart				
B4	B5	B6	B7	B8	B9	B10	B11
B12	....	.....	.....	.....	...		B19
B20	....	.....	.....	.....	...		B27
B28	....	.....	.....	.....	...		B35
B36	....	.....	.....	.....	...		B43
B44	B45	B46	B47	B48	B49	B50	B51

Delete MLPS Label, no PPP encapuslation
MPLS SOP PDU after label modification
MplsMode = 01, PppMode = 0, PduStart = 16
Note: old MPLS Bytes were located in B0-B3
PlBytes = ValidPlBytes - 4

PHY	PlBytes		PduStart				
B0	B1	B6	B7	B8	B9	B10	B11
B12	....	.....	.....	.....	...		B19
B20	....	.....	.....	.....	...		B27
B28	....	.....	.....	.....	...		B35
B36	....	.....	.....	.....	...		B43
B44	B45	B46	B47	B48	B49	B50	B51

Delete MLPS Label, with PPP encapuslation
MPLS SOP PDU after label modification
MplsMode = 01, PppMode = 1, PduStart = 16
Note: old MPLS Bytes were located in B2-B5
PlBytes = ValidPlBytes - 4

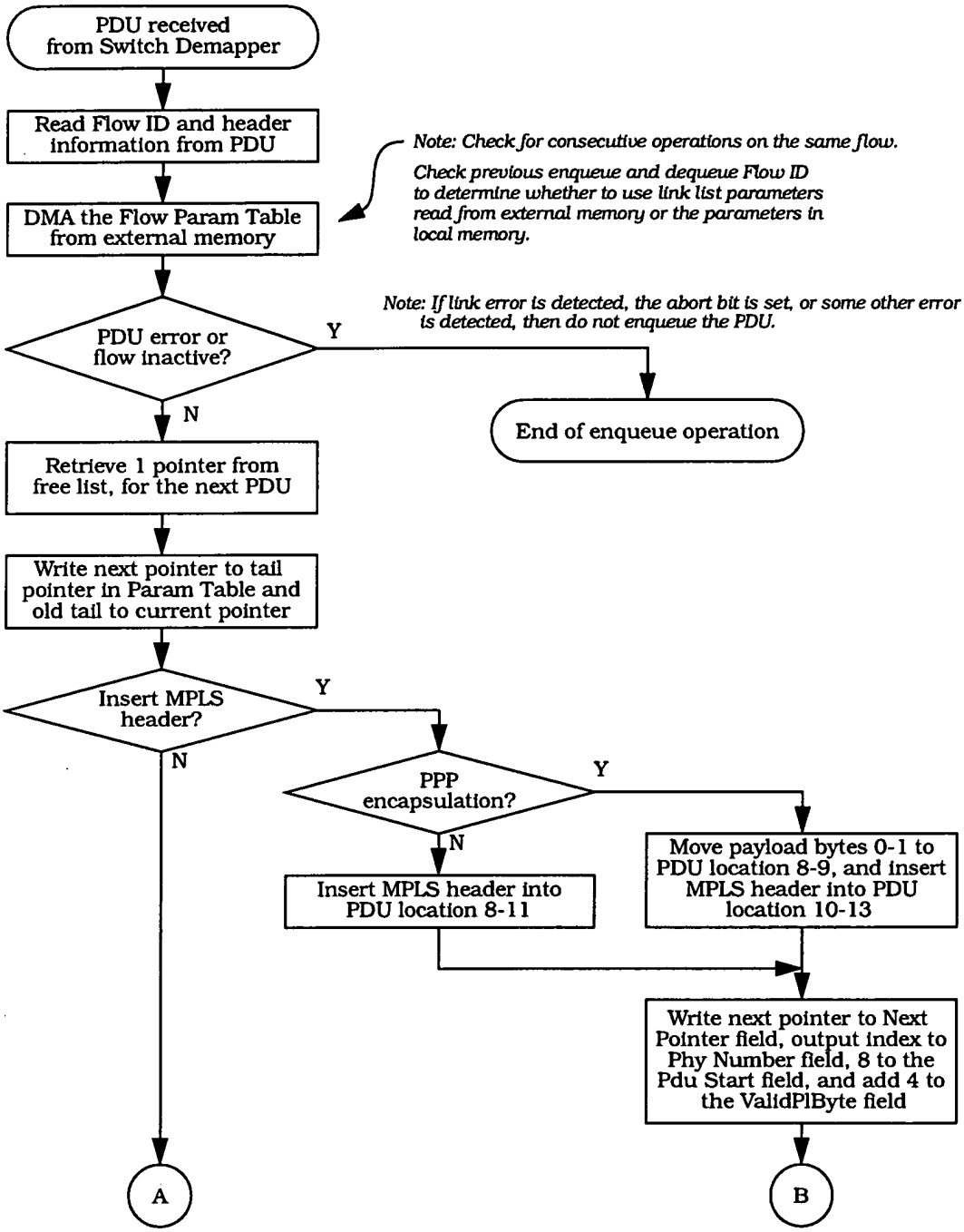
PHY	PlBytes		PduStart				
				M0	M1	M2	M3
B4	B5	B6	B7	B8	B9	B10	B11
B12	....	.....	.....	.....	...		B19
B20	....	.....	.....	.....	...		B27
B28	....	.....	.....	.....	...		B35
B36	....	.....	.....	.....	...		B43
B44	B45	B46	B47	B48	B49	B50	B51

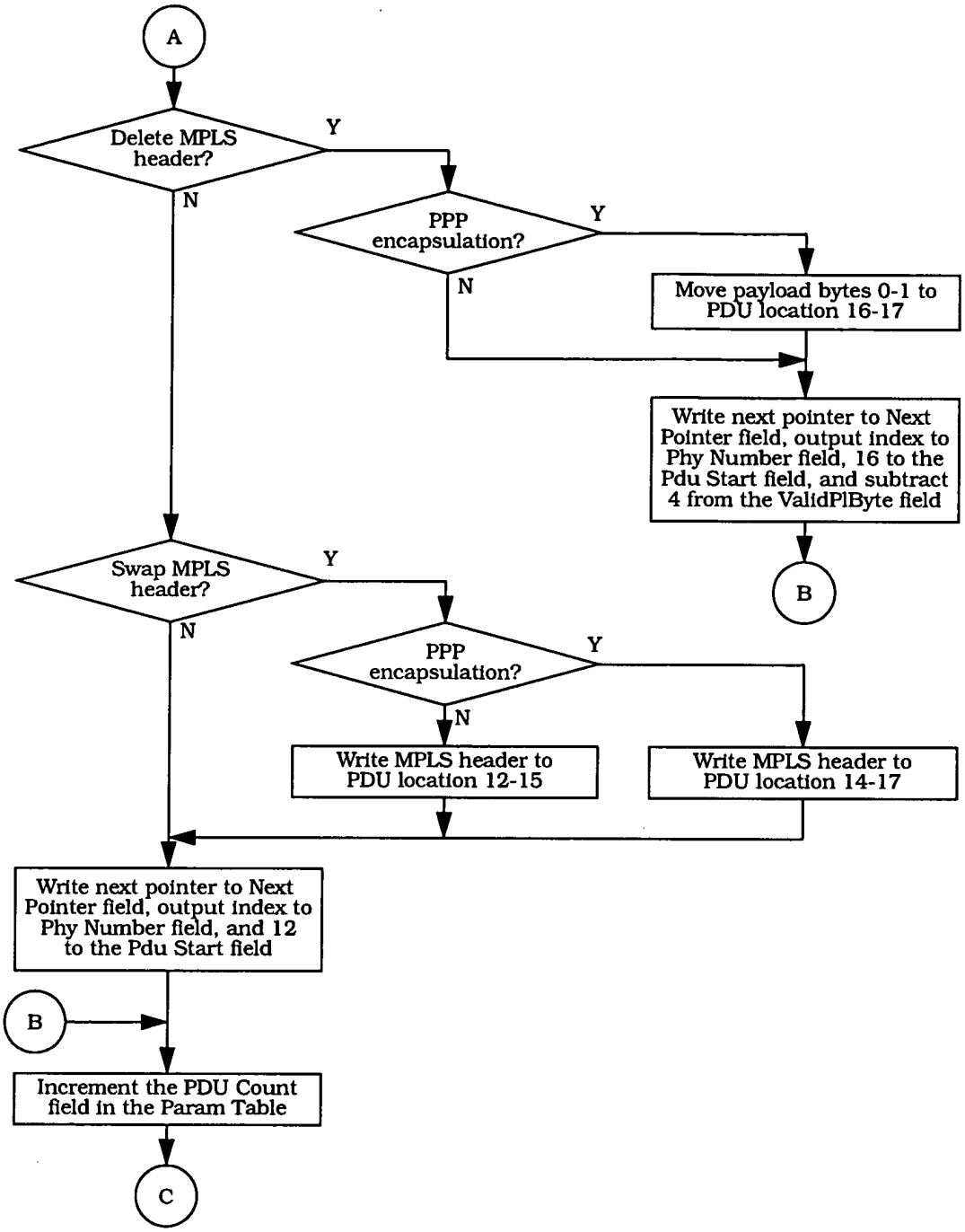
Switch MLPS Label, no PPP encapuslation
MPLS SOP PDU after label modification
MplsMode = 10, PppMode = 0, PduStart = 12
Note: old MPLS Bytes were located in B0-B3
PlBytes = ValidPlBytes

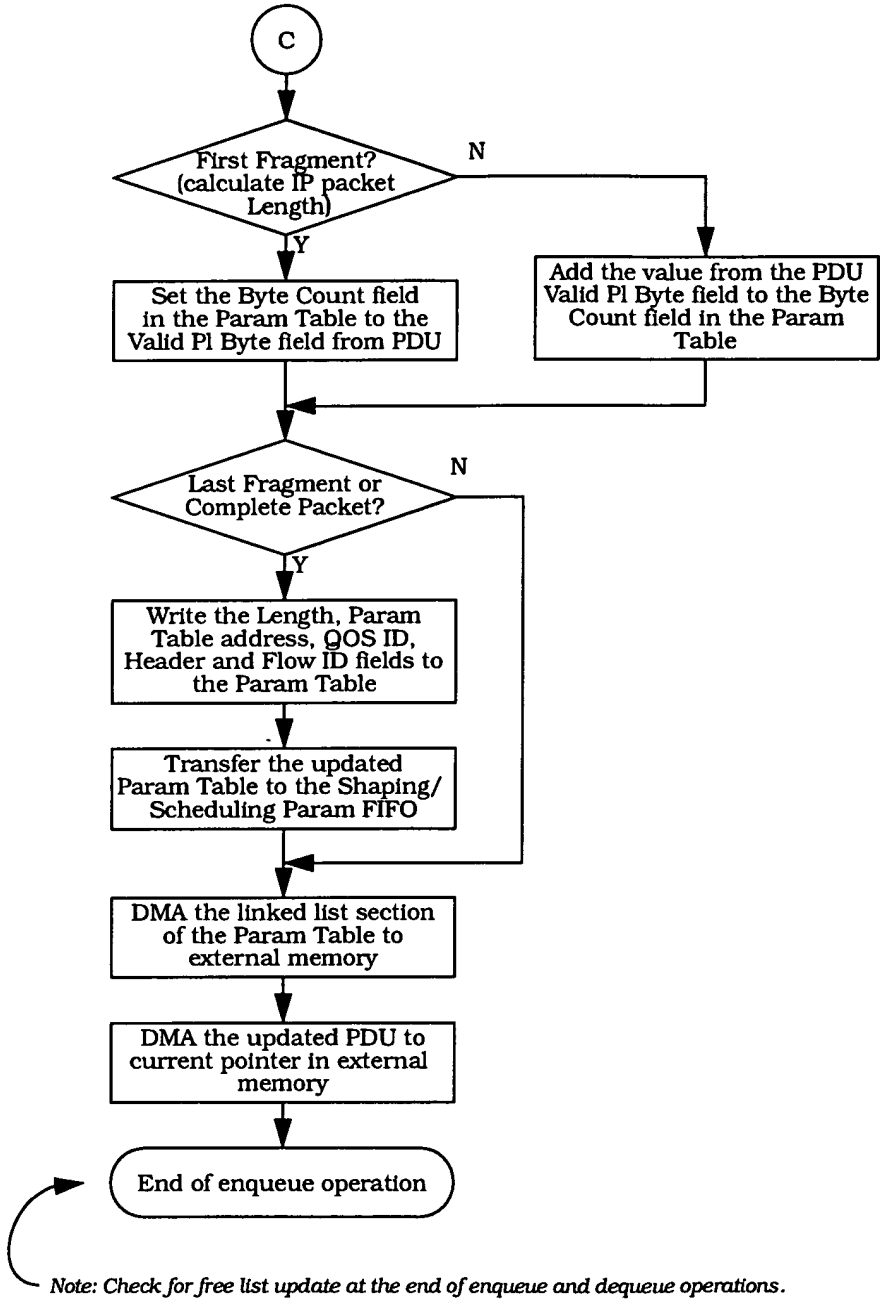
PHY	PlBytes		PduStart				
				B0	B1	M0	M1
M2	M3	B6	B7	B8	B9	B10	B11
B12	....	.....	.....	.....	...		B19
B20	....	.....	.....	.....	...		B27
B28	....	.....	.....	.....	...		B35
B36	....	.....	.....	.....	...		B43
B44	B45	B46	B47	B48	B49	B50	B51

Switch MLPS Label, with PPP encapuslation
MPLS SOP PDU after label modification
MplsMode = 10, PppMode = 1, PduStart = 12
Note: old MPLS Bytes were located in B2-B5
PlBytes = ValidPlBytes

The enqueue process for an MPLS packet PDU is shown in the flow chart below:

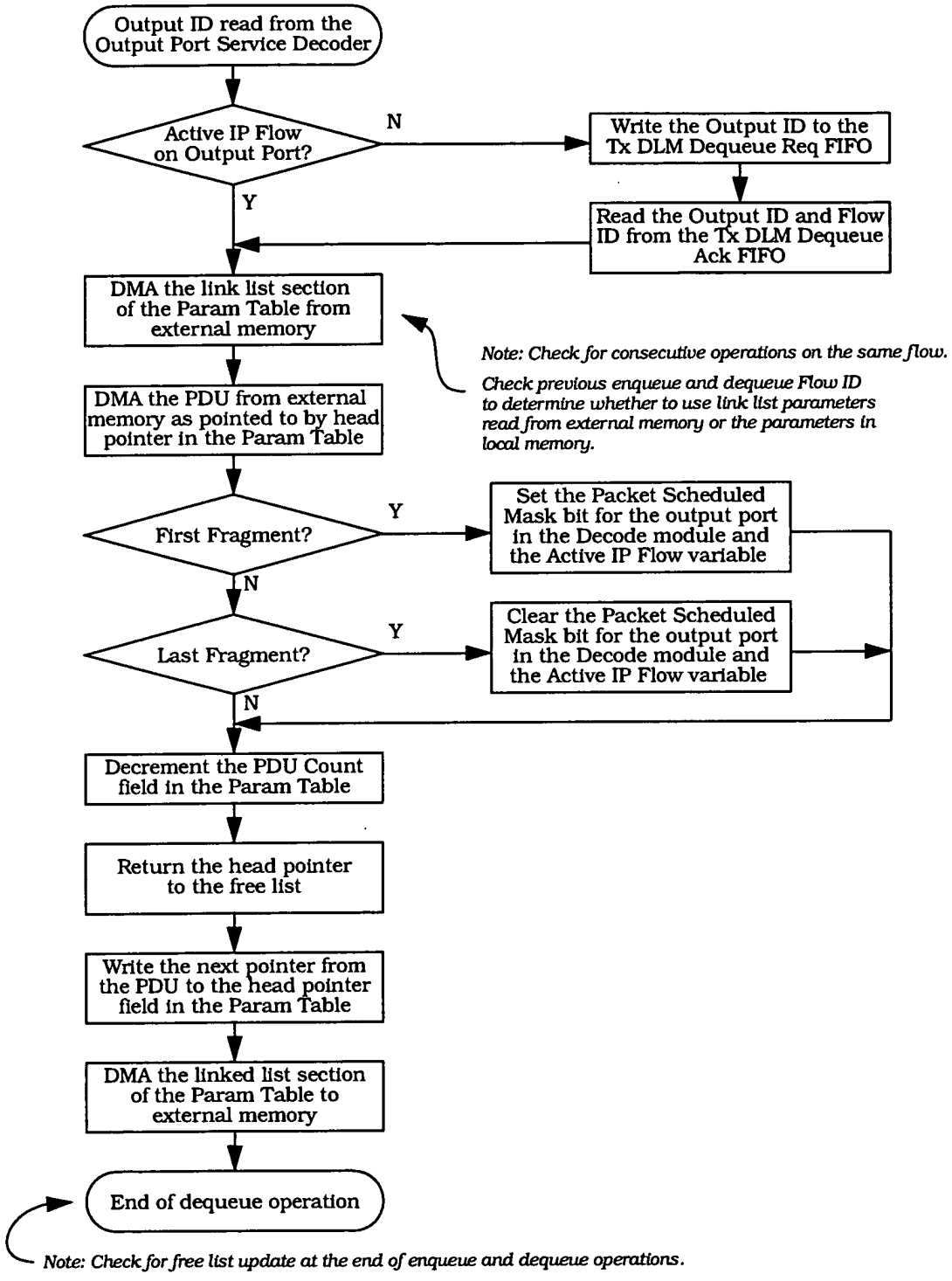






Note: Check for free list update at the end of enqueue and dequeue operations.

The dequeue process for an MPLS packet PDU is shown in the flow chart below:



The control table data structure for an MPLS flow is shown in the table below:

Table 12-9: MPLS Flow Transmit Control Table

6 3	5 5 6 5	4 4 8 7	4 3 0 9	3 3 2 1	2 2 4 3	1 1 6 5	8 7	0	addr offset
Last Finishing Number (dynamic)				IP Byte Count, Cumulative (dynamic)					0x00
Reserved				Reserved					0x08
Reserved				Reserved					0x10
Reserved				Reserved					0x18
Enqueue Head Pointer (dynamic)				PDU Count in Current Packet (dynamic)		Flow Allocation Weight (static)			0x20
Dequeue Head Pointer (dynamic)				MPLS Label (static)					0x28
MiscParams (static)	OutputIndex (static)	Byte Count in Current Packet (dynamic)		Head Pointer (dynamic)					0x30
PDU Count (dynamic)				Tail Pointer (dynamic)					0x38

12.7 Free List Data Structure

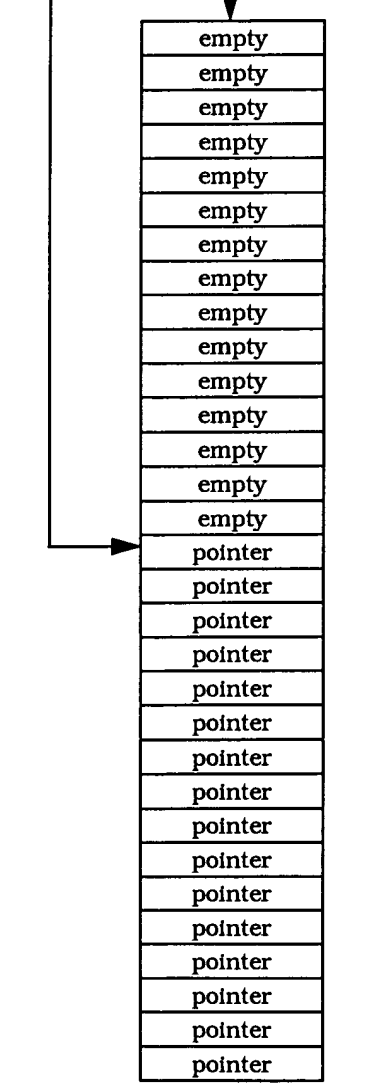
The Free List will be stored in external memory, but a cache will be kept in local data RAM. The idea is to keep 16 full and 16 empty pointers locally. The free pointer data structure is shown

below:

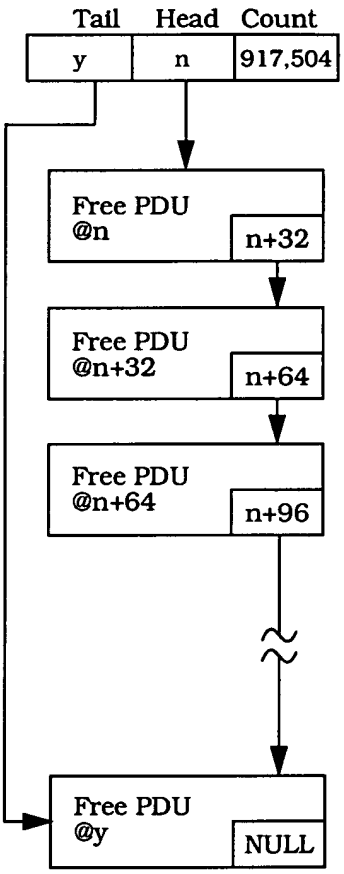
The process will be that enqueues will use pointers from the local cache until there less than 8 pointers left locally. When this threshold is reached, a new pointer will be read from external memory and written into the next local empty location. Likewise, dequeues will write freed up pointers to empty locations in the local cache. When there are less than 8 empty locations left in the local cache, then a pointers will be written back to external memory, freeing up a location in the local cache.

Free List Local Cache at Initialization

Head Index	Head Count	Tail Index	Tail Count
16	16	0	16



Free List External Pointers



Note: Banks 2 and 3 of the external memory are reserved for PDU storage, which allows a maximum of 1M (1,048,576) 64-byte PDUs. At Initialization, the first 128K PDUs are pre-allocated, one per flow, leaving 917,504 PDUs in the free queue.

12.8 Tx DLM Processor FIFO Data Structures

The Tx DLM Processor will communicate with the Tx AIT and the SONET/UTOPIA and Switch Interface hardware modules through FIFOs connected to the Cache Interface (CIF). Address decoding is required to connect the FIFOs to the CIF data bus and generate the FIFO read and write strobes.

13 Transmit Side SONET Interfaces

RESERVED

13.1 Path Overhead Generation

RESERVED

13.2 High Order Pointer Generation

RESERVED

13.3 Transport Overhead Generation

RESERVED

13.3.1 Section Overhead

RESERVED

13.3.2 Line Overhead

RESERVED

13.4 SONET Frame Generation & Scrambling

RESERVED

13.5 Parallel to Serial Conversion

RESERVED

13.6 SPE Generation

RESERVED.

13.6.1 Payload Mapping

RESERVED.

13.6.2 TDM Mapping into SONET

RESERVED

13.6.3 ATM Cell Mapping into SONET

RESERVED

13.6.4 SONET Encapsulation of IP

RESERVED

13.6.5 SPE Routing

RESERVED

13.6.6 Transmit Side Telecom Bus I/F

RESERVED

14 UTOPIA Interface

14.1 UTOPIA Summary

- Level 2. Level 2 Frame Based (UL2+), Level 3 & L3 Frame Based
- Master & Slave
- 96 Supported PHYs in both Master and Slave mode

Number of supported PHYs is limited by the UTOPIA Output side and the amount of buffering required per-PHY. Goal is 192 PHYs because this is the projected number of DSL ports that can be put in one rack. This number is based on a power limitation. The buffering required for support of 192 is too much

- UTOPIA In has one 4-cell FIFO - all PHYs pass through the same FIFO

Since all of the incoming data is destined for the external memory and will be queued with respect to flow ID it doesn't make any sense to exert back pressure on a per-PHY basis. The Data Link Manager (DLM) can stop taking data from the UTOPIA I/F and this would cause the FIFO in the UTOPIA Interface to overflow and the UTOPIA In I/F should set an alarm as this is an error condition. It is a design goal that the RxDLM and the SDRAM controller are able to service the UTOPIA In FIFO fast enough so that the overflow condition will never happen

- Rx DLM must be able to keep this single FIFO close to empty
- Error condition if Input FIFO becomes full - need to set alarm
- UTOPIA Out has a separate FIFO for each supported PHY with a minimum of 3 cells per PHY **(This will probably be 4 cells. We need to evaluate the case of the 104-byte chunk - in this case the buffer only accomodates 2 transfer units. This would only allow 1 input, 1 output and no buffer - just need to verify whether this can inhibit throughput. This is only a concern for the Output direction. In the input direction, the UTOPIA I/F should still assert cell ready to the RxDLM when it has at least 1 cell ready)**

# PHYS	bytes/ cell	cells/ PHY	bytes	bits	cells/ PHY	bytes	bits	cells/ PHY	bytes	bits
1	64	2	128	1024	3	192	1536	4	256	2048
32	64	2	4096	32768	3	6144	49152	4	8192	65536
64	64	2	8192	65536	3	12288	98304	4	16384	131072
96	64	2	12288	98304	3	18432	147456	4	24576	196608
128	64	2	16384	131072	3	24576	196608	4	32768	262144
192	64	2	24576	196608	3	36864	294912	4	49152	393216
256	64	2	32768	262144	3	49152	393216	4	65536	524288

The UTOPIA Interface of the iTPP is an external interface that provides an ATM Forum compliant path to transmit and receive ATM cells and variable length IP Packets. The interface is configurable as Level 2 or Level 3, Master or Slave, and ATM or Packet mode. The data bus width is configurable to be 8-bit, 16-bit or 32-bit. The supported modes are indicated in Table 14-1.

L-2/L-3	M/Sl	ATM/ POS	8/16/32	Supported Y/N	MaxClk (MHz)
L-2	M	ATM	8	Y	104
L-2	M	ATM	16	Y	104
L-2	M	ATM	32	N	
L-2	Sl	ATM	8	Y	104
L-2	Sl	ATM	16	Y	104

L-2/L-3	M/SI	ATM/ POS	8/16/32	Supported Y/N	MaxClk (MHz)
L-2	SI	ATM	32	N	
L-2	M	POS	8	?	
L-2	M	POS	16	?	
L-2	M	POS	32	NA	
L-2	SL	POS	8	?	
L-2	SL	POS	16	?	
L-2	SL	POS	32	NA	
L-3	X	X	8	N	
L-3	X	X	16	N	
L-3	M	ATM	32	Y	104
L-3	M	POS	32	Y	104
L-3	SI	ATM	32	Y	104
L-3	SI	POS	32	Y	104

Table 14-1: UTOPIA Modes

The Master/Slave control bit can be configured differently for the Input direction and the Output direction. All other configuration bits shown in Table 14-1 must be the same for both directions.

The external interfaces to the UTOPIA blocks is specified in the ATM Forum's UTOPIA Level-2 and Level-3 specs. Refer to these specs for more details of the interface as the interface specification is not replicated in this document. The supported and non-supported features are listed in the next sections.

14.2 UTOPIA Level 2 - ATM

In UTOPIA Level-2 mode, the iTPP supports the following:

- Cell-Level Handshaking
- 54-byte cell for 16 bit mode
- 53-byte cell for 8 bit mode
- 56-byte cell in 16 bit mode to allow the addition of routing information.
- 56-byte cell in 8 bit mode to allow the addition of routing information.
- MPHY operation with 1 RxClav and 1 TxClav
- Weighted Round Robin PHY Polling
- 32 PHY addresses in Slave mode
- 32 PHY addresses in master mode
- Back-to-back cell transfer

14.3 UTOPIA Level 2 - Frame Based Interface

In Level 2 Packet mode, the iTPP supports the following enhancements to the UTOPIA Level-2 spec as described in the ATM Forum's XXXXX spec:

- 8 bit data bus?????????
- 16 bit data bus
- packet-level transfer control
- 48 byte Transfer Chunk sizes
- Out of band polling and selection
- 5-bit address bus in the "Output" or "Transmit" direction - from Master to PHY

- Multi PHY Handshaking

14.4 UTOPIA Level 3 - ATM Mode

For UTOPIA Level-3, ATM Mode the iTPP supports the following:

- 32-bit data bus
- 8-bit address bus
- 52-octet cell format as shown in Table 2.1 of the UTOPIA L3 spec, dated November 1999
- 56-octet cell format as shown in Table 2.2 of the UTOPIA L3 spec, dated November 1999
- control of up to 96 PHYs in Master Mode
- Weighted Round Robin PHY Polling
- response to up to 96 PHY addresses in slave mode
- Multi-PHY Operation with 1 TxClav and 1 RxClav Signal
- Back-to-back cell transfers

14.5 UTOPIA Level 3 - Frame Based Interface

In Packet mode, the iTPP supports the following enhancements to the UTOPIA Level-3 spec as described in the ATM Forum's Frame-based ATM Interface spec:

- 32-bit data bus
- packet-level transfer control
- Transfer Chunk sizes
  - 52 bytes
  - (<52) 48 bytes - we will pad 4 bytes through the switch and notify the far-end PP
  - (>52) 104 bytes - anything greater than 52 - and not a multiple will break the single input FIFO idea as we would be storing partial packets and partial headers and we would need to wait for the rest of the header before we could forward it to the IPF
- polled status
- 8-bit address bus in the "Output" or "Transmit" direction - from Master to PHY

14.6 UTOPIA I/F in iTAP iTPP

The UTOPIA interface is high-lighted in the full iTPP block diagram shown in 14-2. The UTOPIA blocks are labelled as "Input" and "Output" to correspond to the direction of data flow. The labels "Receive" and "Transmit" are used in the UTOPIA specs where "Receive" indicates data flowing from the PHY to the Master and "Transmit" indicates data flowing from the Master to the PHY. Since the iTPP can be configured as Master or Slave, it would be confusing to label them as Rx and Tx as these labels would change depending on the configured mode.

The direction of some of the UTOPIA control signals (Address, Enables, Clavs...) changes depending on the Master Slave configuration of the iTPP. Any signal that changes direction in this way is implemented as a bi-directional pin with the Master/Slave configuration bit controlling the input/output selection.

14-2 is a conceptual block diagram. The Architectural block diagram which describes the actual implementation is described in later sections.

UTOPIA Level-2 requires LVTTL buffers. UTOPIA Level-3 requires HSTL I/O buffers. This may require us to duplicate the chip level I/O signals that are shared between Level-2 and Level-3.

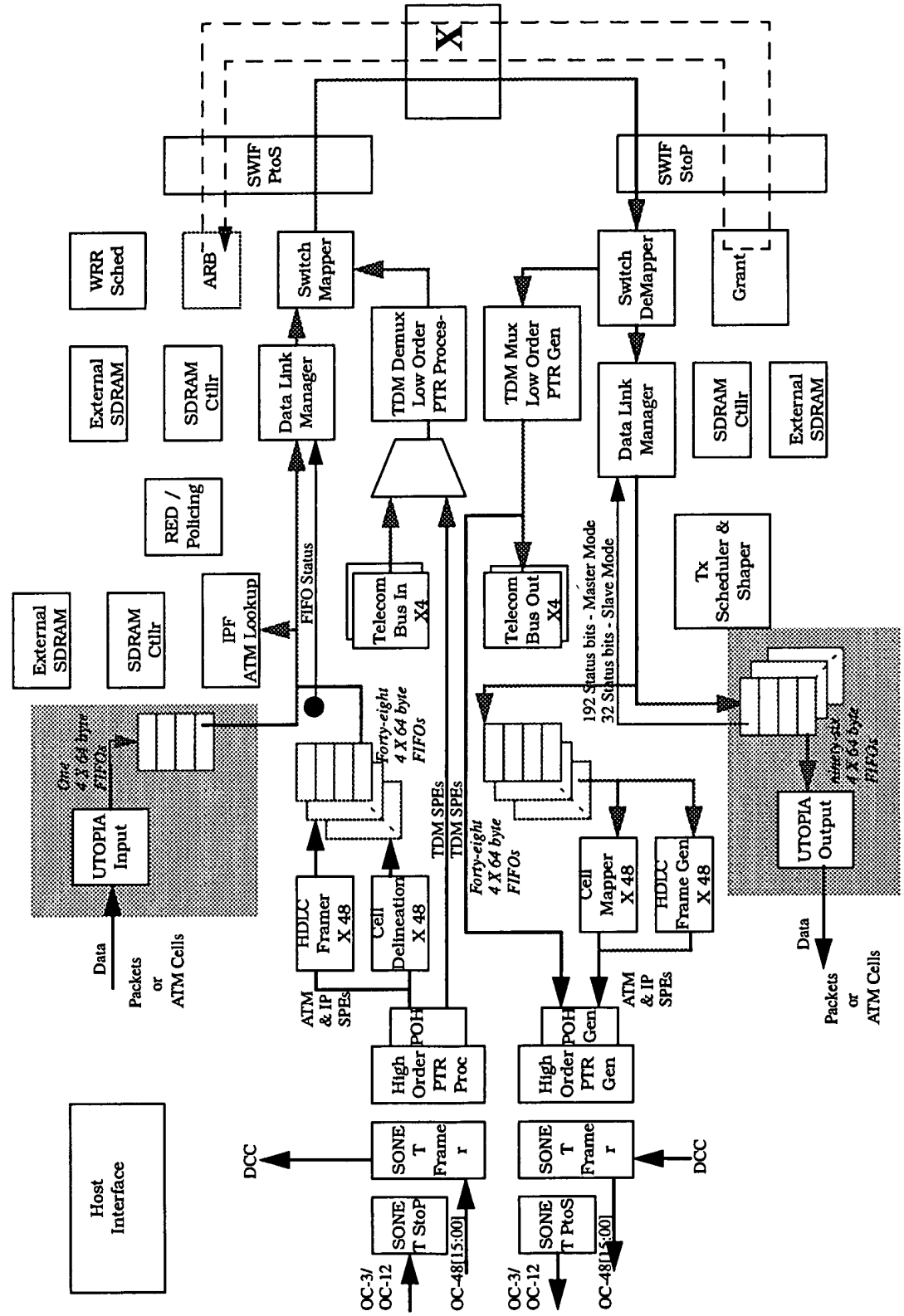


Figure 14-2: fTPP Block DIagram

93

14.7 UTOPIA Input Interface

The Block diagram and I/O are shown in Figure 14-3.

In **ATM mode**, the UTOPIA In block can operate in two modes - Normal and extended byte (XT_byte) mode. In Normal mode, this block buffers complete 52-byte cells (no HEC). These cells will eventually be transferred to external Rx memory under control of another block. The UTOPIA In block supports a 53-byte cell in 8-bit data bus mode in which case the HEC (UDF) byte will be dropped. The UTOPIA In block supports a 54-byte cell in 16-bit data bus mode in which case the two UDF bytes will be dropped. This UTOPIA block indicates when it has at least one complete cell in its buffer by asserting the Cell Ready signal.

In extended byte mode, the UTOPIA interface supports a 56-byte cell in 8-bit data bus mode and a 56-byte cell in 16-bit data bus mode. This mode allows a user to attach his own router/ classifier to the UTOPIA Input Interface and bypass the internal ATM Lookup processor. In this mode, four extra bytes are inserted into the 4 UDF bytes of the ATM cell. These 4 extra bytes contain the information that is needed to create the ATM cell descriptor that would normally be output by the ATM Lookup processor. This mode is enabled through the Rx_XT_Byte_Mode configuration bit. The data structure for this 56-byte cell is referenced in Table 14-12.

In **Packet mode**, this block inputs and buffers chunks of packets destined for the external Rx memory. As in ATM mode, transfer to the external memory is under full control of a different block. This "chunking" is actually segmenting the incoming packet into an iTAP sized PDU. The packet chunk size is the number of bytes transferred across the UTOPIA interface and is programmable to one of three values: 48, 52 or 104 bytes. The payload area of an iTAP Switch Mapped data cell is 52 bytes. Once a packet chunk has begun to be transmitted across the UTOPIA interface, it must be completely transferred before another packet chunk can begin. This UTOPIA block indicates when it has a packet chunk ready for transfer by asserting the Cell Ready signal. Chunks of packets from different PHYs can be interleaved on the UTOPIA interface, but a chunk is always transferred completely before another chunk can start. The UTOPIA interface detects the final bytes of a variable length packet by sampling the End of Packet Signal. The transfer of variable length packets into the iTPP typically results in a partial chunk (less than the programmed number of bytes) being used for the final bytes of a packet. The UTOPIA Input block indicates the last byte of a packet by asserting a similar End of Packet signal to the next block inside the iTPP as the last word is clocked out of the input FIFO.

If the Rx_XT_Byte_Mode configuration bit is set in Packet mode, then 4 bytes are added to the beginning of the first chunk of an IP Packet. These 4 additional bytes are only added to the first chunk of a packet - all subsequent chunks revert back to the programmed chunk size. The first 4 bytes of the first packet chunk are assumed to be routing / classification information that has been calculated by an external 3rd party IP forwarding / classification engine. These bytes will be substituted in place of the output of the IP forwarding and Classification engine.

In both ATM and Packet mode, the UTOPIA Input block has a single 4-cell (implemented as 64 bytes) FIFO which is used for clock separation and data buffering. The first location for each cell or packet chunk contains the PHY ID number. The data interface from the UTOPIA Input block into the iTPP is through this FIFO. There is also a separate Header FIFO that is used to hold ATM and IP header information. Up to four headers can be stored. The control interface from the UTOPIA Input block into the iTPP is through this FIFO. Both of these FIFOs are inside the UTOPIA Input block. The Read control signals for these FIFOs are inputs to the block and should be treated as asynchronous to the UTOPIA Input timing.

The back-pressure to UTOPIA Input block is the rate at which data is removed from the transfer FIFO by the Data Link Manager. If the DLM can not keep up with the rate of the incoming traffic, the buffer will fill and this is an error condition. The UTOPIA In block sets an alarm to indicate a full buffer.

This function is no longer in this block. It needs to be done on the SONET interfaces also, so it will be done in the block that creates the descriptor as all of the SONET traffic and the UTOPIA traffic go through it.

The UTOPIA Master interface services the ninety-six FIFO buffers in a weighted round robin manner as described in Section 14.9.

14.7.1 UTOPIA Input Block I/O

- **Inputs**
  1. All inputs required to support UTOPIA Level 2 & 3 - listed in Table 14-8.
  2. Mode configuration and enable signals
     - L2/L3 - '0' = Level 2 mode, '1' = Level 3 mode
     - ATM/Packet = '0' = ATM mode, '1' = Packet mode
     - Master/Slave - '0' = Master, '1' = Slave - could be different for Input side and Output side
     - PHY Enables - '1' = Enabled, '0' - not enabled. Separate bit for each PHY. Slave does not respond to polls that are not internally enabled - it leaves its CLAV tri-stated when it receives a poll for a not enabled PHY. The Master could poll a disabled PHY, but it will ignore the response of the PHY if the enable bit is not set.
     - Chunk Size = 48, 52 or 104
     - Rx_XT_Byte_Mode - '0' = normal mode, '1' = extended byte mode. When this bit is set in ATM mode, 4 extra bytes have been inserted into the 4 UDF bytes of each incoming ATM cell. The ATM cell transfer size is 56 bytes in 8 bit L2 mode, 56 bytes in 16 bit L2 mode and 56 bytes in 32-bit L3 mode. When this bit is set in IP mode, 4 extra bytes are prepended to the first chunk of an IP Packet. The four extra bytes are placed into the 'T' byte locations shown in Figure 14-3.
  3. Header FIFO Read control signals
  4. Data FIFO Read control signals
- **Outputs**
  1. All outputs required to support UTOPIA Level 2 & 3 - listed in Table 14-8.
  2. Data FIFO status
  3. Header FIFO status.
  4. Start of Cell / Packet - active during the transfer of word #1 which includes the PHY number.
  5. End of Cell / Packet
  6. Abort Indication - forces packet indicated by ID bits to be discarded. **(IP only)**
  7. Status signals and counters

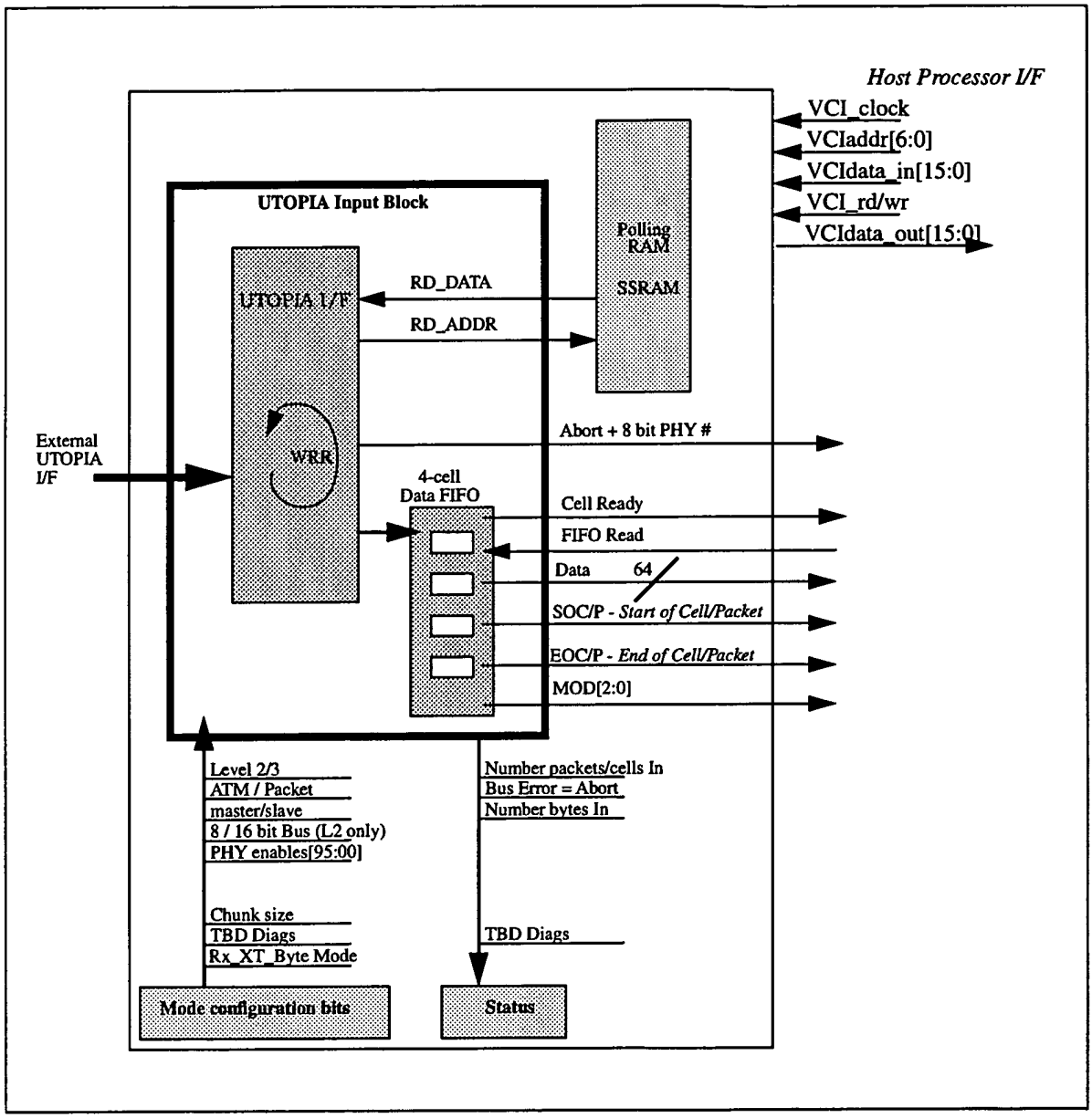


Figure 14-3: UTOPIA Input Block Diagram

The data structure shown in Figure 14-4 shows the data structure that is clocked out of the Input side PDU FIFO. The only variation from this data structure is in packet mode when the chunk size is programmed to 48 bytes. In this 48-byte mode, the last 4 bytes of the payload (B48 - B51) will be forced to all 0's. In packet mode, the UTOPIA Input block will generate the EOP signal to indicate the word that contains the last byte of the packet and will also generate the MOD bits to indicate tbe number of bytes that are valid in this last word. A value of '000' with the EOP bit set indicates that the last byte is in the first location of this word and that the other 7 bytes in the word are not valid. '001' identifies the last byte as being in the second byte and so on.

The PHY # is needed in IP mode by the RxDLM to identify the link for an ongoing packet receipt as the RxDLM only gets a descriptor for the first chunk of a packet

63 0

PHY#				T1	T2	T3	T4
B0	B1	B2	B3	B4	B5	B6	B7
B8	....	.....	.....	.....	...		B15
B16	....	.....	.....	.....	...		B23
B24	....	.....	.....	.....	...		B31
B32	....	.....	.....	.....	...		B39
B40	....	.....	.....	.....	...		B47
B48	B49	B49	B51				

In Packet mode, the "T" bytes are only valid for the fist chunk of a packet.

Figure 14-4: Output Data Structure for UTOPIA Input PDU FIFO

The Input FIFO must store the EOP information temporarily while it is waiting for the DLM to take the PDU. The FIFO structure is actually 68 bits wide as shown in Figure 14-5 with 64 bits being used to store the PDU data and the extra 4 bits being used to store the EOP and MOD bits. This eases the task of generating the EOP and MOD bits as they are clocked directly out of the FIFO onto the EOP and MOD signals.

67 66 65 64 63.....0

EOP	MOD2	MOD1	MOD0	PHY#				T1	T2	T3	T4
EOP	MOD2	MOD1	MOD0	B0	B1	B2	B3	B4	B5	B6	B7
EOP	MOD2	MOD1	MOD0	B8	....	.....	.....	.....	...		B15
EOP	MOD2	MOD1	MOD0	B16	....	.....	.....	.....	...		B23
EOP	MOD2	MOD1	MOD0	B24	....	.....	.....	.....	...		B31
EOP	MOD2	MOD1	MOD0	B32	....	.....	.....	.....	...		B39
EOP	MOD2	MOD1	MOD0	B40	....	.....	.....	.....	...		B47
EOP	MOD2	MOD1	MOD0	B48	B49	B49	B51				

Figure 14-5: Input FIFO Data Structure

The HDR bytes are shown in the data structures referred to in Table 12-8.
The "T" bytes are taken from the UDF bytes of the ATM cell

PHY #	HDR1	HDR2	HDR3	HDR4
	T1	T2	T3	T4

Figure 14-6: UTOPIA Input Header FIFO data Structure - ATM mode

Under Construction

Figure 14-7: UTOPIA Input Header FIFO data Structure - Packet mode

Signal	Direction	L2 Master	L2 Slave	L3 ATM Master	L3 ATM Slave	L3 Frame-based Master	L3 Frame-based Slave
IADD[7:0]	In/Out *In-slave* *Out-mstr*	RxAddr[4:0]	TxAddr[4:0]	RxAddr[7:0]	TxAddr[7:0]	*in-band addr*	TADR[?:?]
IDAT[31:0]	Input	RxData[15:0]	TxData[15:0]	RxData[31:0]	TxData[31:0]	RDAT[31:0]	TDAT[31:0]
ISOC/P	Input	RxSOC	TxSOC	RxSOC	TxSOC	RSOP	TSOP
IEOP	Input	-	-	-	-	REOP	TEOP
IENB*	In/Out *In-slave* *Out-mstr*	RxEnb*	TxEnb*	RxEnb*	TxEnb*	RENB	TENB
ICLAV	In/Out *In-mstr* *Out-slave*	RxClav	TxClav	RxClav[0]	TxClav[0]	-	PTPA
ICLK	Input	RxClk	TxClk	RxClk	TxClk	RFCLK	TFCLK
IERR	Input	-	-	-	-	RERR	TERR
IMOD[1:0]	Input	-	-	-	-	RMOD[1:0]	TMOD[1:0]
ISX	Input	-	-	-	-	RSX	TSX
IVAL	Input	-	-	-	-	RVAL	-

Table 14-8: UTOPIA Input Interface Signals

14.8 UTOPIA Output Interface

The Block diagram and I/O are shown in Figure 14-9.

In **ATM mode**, the UTOPIA Output block buffers complete 52/3/4/6-byte cells which will eventually be transferred onto the external UTOPIA interface. This block makes the cell available to the UTOPIA interface only after the entire cell is received into its FIFO. For 52, 53 and 54 byte cells, the UTOPIA Output block will receive 52 bytes into its FIFO and the UTOPIA Output block will insert 0's into the UDF byte locations. For 56 byte cell transfer mode, the UTOPIA Output block will receive all 56 bytes into its FIFO. In this 56 byte transfer mode, the data structure referred to in Table 14-12 is used. The UTOPIA Output block can support up to 96 PHY devices in Slave mode and in Master mode. The goal of the iTPP is to keep the Output UTOPIA buffers full and ready to output a cell. In order to do this, the iTPP must maintain a separate buffer for each supported PHY address.

In **Packet mode**, the UTOPIA Ouput block buffers chunks of packets destined for the external UTOPIA Interface. These chunks are clocked onto the UTOPIA data bus as one contiguous packet. Once a packet begins to be transmitted out the UTOPIA interface, it is the responsibility of the Data Link Manager in the iTPP to keep the buffer sufficiently full so that there are no gaps on the UTOPIA data bus. Once a packet chunk has begun to be transmitted across the UTOPIA interface, it must be completely transferred before another chunk can begin. The UTOPIA interface terminates the transfer of the variable length packets by asserting the End of Packet Signal. The UTOPIA Output block indicates the last word of a packet by asserting the End of Packet signal. The packet chunk size used for FIFO statusing is programmable and can be set to 48, 52 or 104 bytes.

In both ATM and Packet mode, the UTOPIA Output block has ninety-six 4-cell (64 bytes each) FIFOs which are used for clock separation and data buffering. There is per-PHY back-pressure. The data interface into the UTOPIA Output block from the iTPP is through this FIFO. This FIFO is inside the UTOPIA Output block. The Write control signals for these FIFOs are inputs to the block and should be treated as asynchronous to the UTOPIA Input timing.

The UTOPIA Master interface services the ninety-six FIFO buffers in a weighted round robin manner as described in Section 14.9.

Whenever a PHY is in a disabled state, the internal interface must report "not ready" for that PHY. Whenever one of the PHY enable signals toggles to off, the UTOPIA block must be able to remove from its buffers any packets or cells for the newly disabled PHY.

14.8.1 UTOPIA Output Block I/O

- **Inputs**
  1. All inputs required to support UTOPIA Level 2 & 3 - listed in Table 14-11.
  2. Mode configuration and enable signals - Same as in Input Direction - can be the same control signals except for the Master/Slave bit. The Master/Slave configuration can be different for Input and Output
     - Master/Slave - could be different for Input side and Output side
     - L2/L3
     - ATM/Packet
     - PHY Enables - '1' = Enabled, '0' - not enabled. Separate bit for each PHY. Slave does not respond to polls that are not internally enabled - it leaves its CLAV tri-stated when it receives a poll for a not enabled PHY. The Master could poll a disabled PHY, but it will ignore the response of the PHY if the enable bit is not set. The UTOPIA Output block will not accept cells or packets from the DLM for disabled PHYs.
     - Chunk Size = 48, 52 or 104
     - Tx_XT_Byte_Mode - '0' = normal mode, '1' = extended byte mode. When this bit is set in ATM mode, 4 extra bytes have been added to each incoming ATM cell. The ATM cell transfer size is 56 bytes in 8 bit L2 mode, 56 bytes in 16 bit L2 mode and 56 bytes in 32-bit L3 mode. The four extra bytes are taken from the 'T' byte locations shown in Figure 14-10.
  3. Data FIFO Write control signals
  4. Start of Cell / Packet
  5. End of Cell / Packet

6. Abort Indication - Frame based mode only

- **Outputs**
  1. All outputs required to support UTOPIA Level 2 & 3 - listed in Table 14-11.
  2. Data FIFO status for 96 FIFOs
  3. Status signals and counters

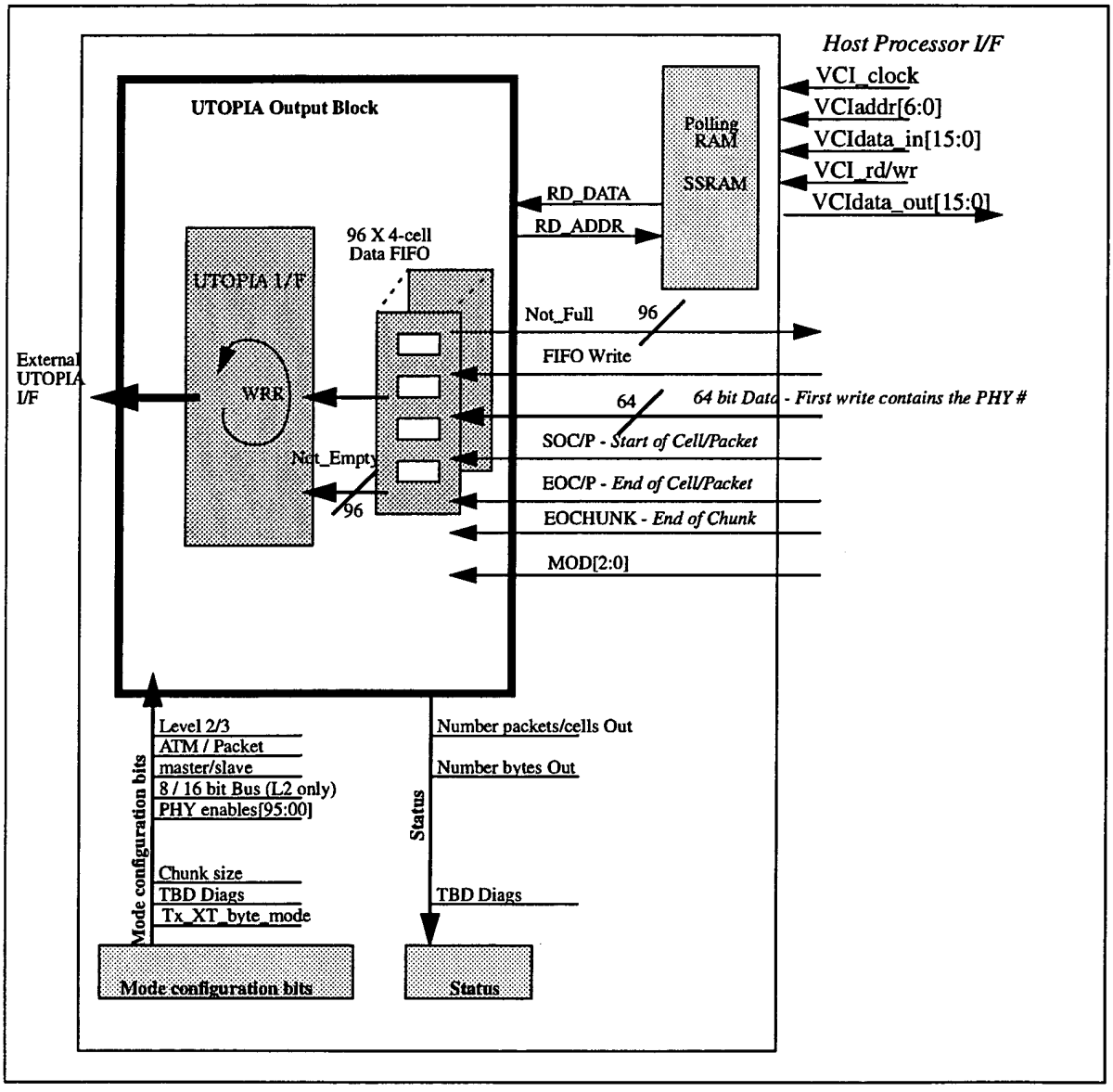


Figure 14-9: UTOPIA Output Block Diagram

The data structure shown in Figure 14-10 shows the data structure that is clocked into the Output side PDU FIFO. The only variation from this data structure is in packet mode when the chunk

size is programmed to 48 bytes. In this 48-byte mode, the last 4 bytes of the payload (B48 - B51) will be forced to all 0's. In packet mode, the UTOPIA Output block will receive the EOP signal to indicate the word that contains the last byte of the packet and will also receive the MOD bits to indicate tbe number of bytes that are valid in this last word. A value of '000' with the EOP bit set indicates that the last byte is in the first location of this word and that the other 7 bytes in the word are not valid. '001' identifies the last byte as being in the second byte and so on.

The PHY # is used in both IP & ATM modes by the UTOPIA interface to identify the buffer that the PDU should be put in.

63 0

PHY				T1	T2	T3	T4
B0	B1	B2	B3	B4	B5	B6	B7
B8	....	.....	.....	.....	...		B15
B16	....	.....	.....	.....	...		B23
B24	....	.....	.....	.....	...		B31
B32	....	.....	.....	.....	...		B39
B40	....	.....	.....	.....	...		B47
B48	B48	B49	B51				

If extended byte mode is enabled in the output direction, the 4 "T" bytes will be output onto the UTOPIA bus in the 4 UDF bytes of the ATM cell or as the first 4 bytes of an IP packet.

In Packet mode, the "T" bytes can only be valid in the first chunk of an IP packet. These byte locations are undefined for the non-first PDUs.

Figure 14-10: Input Data Structure for UTOPIA Output PDU FIFO

The Output FIFO must store the EOP information temporarily while it is waiting for the UTOPIA interface to take the PDU. The FIFO structure is the same as in the Input Block as shown in Figure .

Signal	Direction	L2 Master	L2 Slave	L3 ATM Master	L3 ATM Slave	L3 Frame-based Master	L3 Frame-based Slave
OADD[7:0]	In/Out *In-slave Out-mstr*	TxAddr[4:0]	RxAddr[4:0]	TxAddr[7:0]	RxAddr[7:0]	TADR[?:?]	*in-band addr*
ODAT[31:0]	Output	TxData[15:0]	RxData[15:0]	TxData[31:0]	RxData[31:0]	TDAT[31:0]	RDAT[31:0]
OSOC/P	Output	TxSOC	RxSOC	TxSOC	RxSOC	TSOP	RSOP
OEOP	Output	-	-	-	-	TEOP	REOP
OENB*	In/Out *In-slave Out-mstr*	TxEnb*	RxEnb*	TxEnb*	RxEnb*	TENB	RENB
OCLAV	In/Out *In-mstr Out-slave*	TxClav	RxClav	TxClav[0]	RxClav[0]	PTPA	-
OCLK	Input	TxClk	RxClk	TxClk	RxClk	TFCLK	RFCLK
OERR	Output	-	-	-	-	TERR	RERR
OMOD[1:0]	Output	-	-	-	-	TMOD[1:0]	RMOD[1:0]
OSX	Output	-	-	-	-	TSX	RSX
OVAL	Output	-	-	-	-	-	RVAL

Table 14-11: UTOPIA Output Interface Signals

14.9 Weighted Round Robin Polling

RESERVED.

14.10 Configuration and Alarms

Table 14-12 in this section maps the configuration bit combinations to the data transfer size across the UTOPIA Interface. The Data Structure column in the table indicates the reference for the data structure for each mode. In ATM XT_Byte mode, the 4 UDF bytes in the Data Structure are used to carry the added routing information. In IP XT_Byte mode, the first 4 bytes of the first chunk of the packet carry the added routing information.

L2/L3	ATM/ Packet	8/16	XT_byte mode	Chunk Size	Transfer Size(bytes)	Transfer Data Structure
L2	ATM	8	0	X	53	UTOPIA L1, V2.01, Fig 2
L2	ATM	8	1	X	56	UTOPIA L3, Nov '99,Table 2.2
L2	ATM	16	0	X	54	UTOPIA L1, V2.01, Fig 8
L2	ATM	16	1	X	56	UTOPIA L3, Nov '99,Table 2.2
L3	ATM	X	0	X	52	UTOPIA L3, Nov '99,Table 2.1
L3	ATM	X	1	X	56	UTOPIA L3, Nov '99,Table 2.2
L3	Packet	X	0	48	48	Frame Based ATM Interface (Extension to UTOPIA L3), Feb 2000 Fig 5.1 - N=48 not 109
L3	Packet	X	1	48	1st=52 others=48	others = same as above. 1st = same as above except N=52 & Bytes 1-4 carry proprietary routing info
L3	Packet	X	0	52	52	Frame Based ATM Interface (Extension to UTOPIA L3), Feb 2000 Fig 5.1 - N=52 not 109
L3	Packet	X	1	52	1st=56 others=52	others = same as above. 1st = same as above except N=56 & Bytes 1-4 carry proprietary routing info
L3	Packet	X	0	104	104	Frame Based ATM Interface (Extension to UTOPIA L3), Feb 2000 Fig 5.1 - N=104 not 109
L3	Packet	X	1	104	1st=108 others=104	others = same as above. 1st = same as above except N=108 & Bytes 1-4 carry proprietary routing info

Table 14-12: UTOPIA Transfer Size Configuration

An Alarm is required if the UTOPIA Output Block receives a PDU for a PHY that is disabled. This will be an indication of a misconfiguration either in this Output Port Processor or in the originating Input Port Processor. If the UTOPIA Ouput block receives a PDU for a disabled PHY, the PDU is dropped and the alarm will indicate which disabled PHY was sent a PDU. This requires that the PHY Enable information be synchronized to both the UTOPIA timing domain and also to the DLM timing domain. The easiest way to have these config bits in both domains is to have the Host configure them into one domain and then double sample them into the other. The implementor of this block can consider other more creative solutions to this problem.

15 SONET DCC

15.1 Tx Section & Line DCC

THIS SECTION SHOULD/WILL BE TRANSFERRED TO THE Rx AND Tx SONET SECTIONS

The Section and Line DCC bytes are supported in the iTAP Service Processor. The Tx SONET interface allows a micro processor to insert messages that are less than or equal to 256 bytes into the Section DCC bytes (D1-D3) and Line DCC bytes(D4-D12) of the outgoing SONET Signals. The iTAP Service Processor inserts the DCC messages into the DCC bytes associated with STS-1 #1 of the OC-3(c), OC-12(c) and OC-48(c) ports or into the first DCC bytes of an SDH port. Some of the other DCC byte locations have been designated as "NU", National Use or "MDB", Media Dependent Bytes. These NU and MDB bytes as well as the undefined DCC byte locations are accessible through the processor interface, but are not part of the DCC support described here.

As shown in Figure 15-1, the DCC is used to provide host processors at different points in a SONET network with an in-band communication channel. The iTAP Service Processor encapsulates the DCC messages in HDLC. The HDLC format is shown in Figure 15-2. The Service Processor adds the HDLC flags and the 16 or 32 bit FCS fields. All other fields must be loaded through the processor interface.

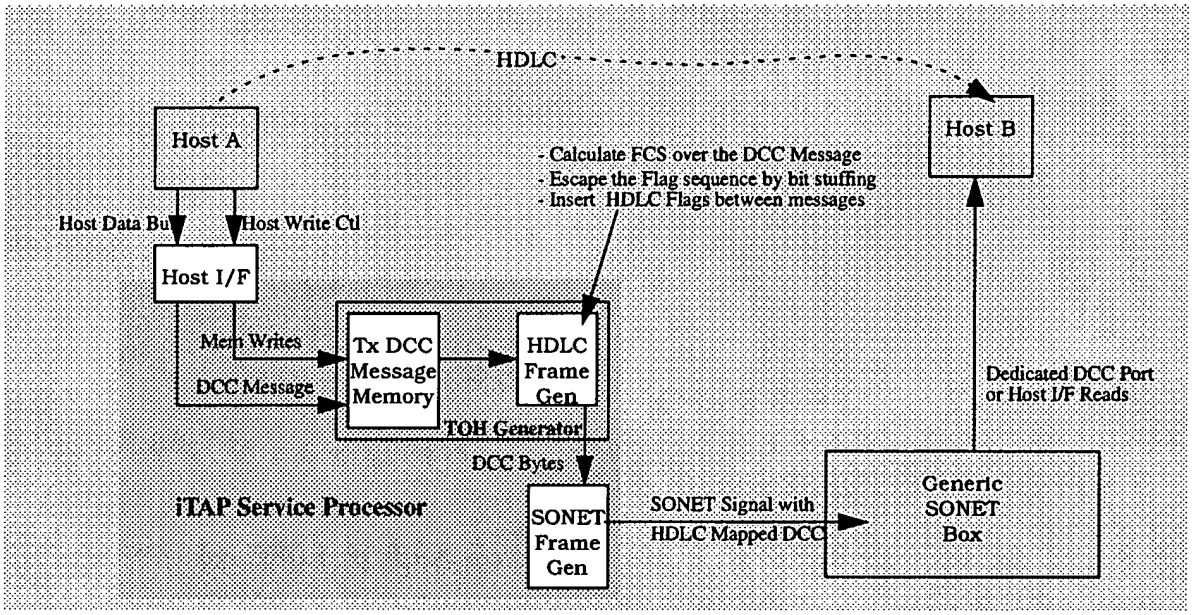


Figure 15-1: Tx DCC

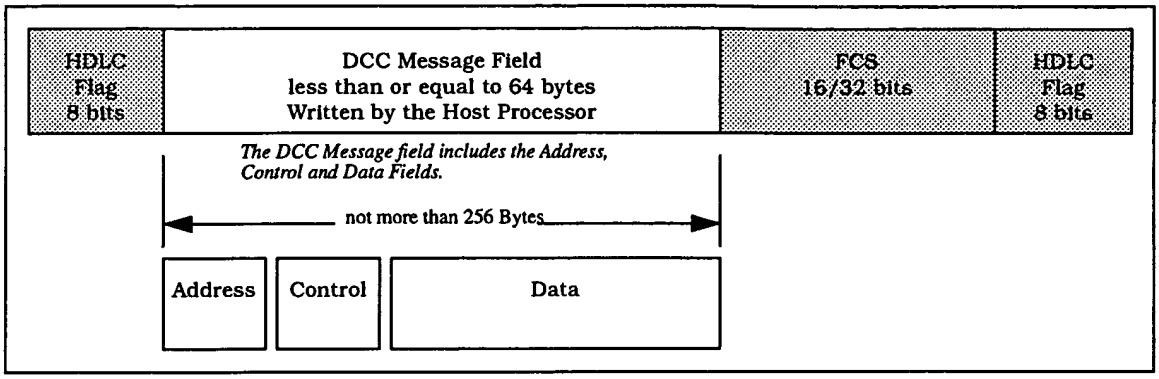


Figure 15-2: HDLC Format

The TOH generator of the Tx SONET interface has a read/write processor interface through which the processor is able to load DCC messages into a mesage RAM. Two 256-byte sections of memory are allocated to each DCC Channel. Two messages for each Section DCC and Line DCC of each of the four SONET interfaces (the OC-48 interface will use one of these four) equates to storage for sixteen 256-byte messages. These memory locations are treated as 32-bit registers so that they can be separately accessed by the processor. After loading in a message, the processor must write the length of the message and a control bit that triggers the TOH generator to mux the particular HDLC encapsulated message into the outgoing DCC byte locations. The message length and the control bit are stored in a separate memory from the message itself. A separate DCC control register is allocated to each of the 16 messages so that either of the two stored messages for each channel can be sent out in any order.

The HDLC protocol is described in rfc1662. The Service Processor supports Bit-Stuffed Framing as described in Section 5. The Service Processor does NOT support Octet-stuffed framing.

The FCS is calculated on-the-fly as the DCC message is muxed into the outgoing SONET signal. "0" bit insertion to support Transparency is also performed on-the-fly - after FCS computation. Performing "0" bit insertion on the way out of the message buffer results in a complex output circuit where words read out of the buffer could be shifted bit-by-bit and thereby not directly muliplexed into the outgoing SONET stream. Figure 15-3 shows a proposal for the implementation of

the '0' bit insertion.

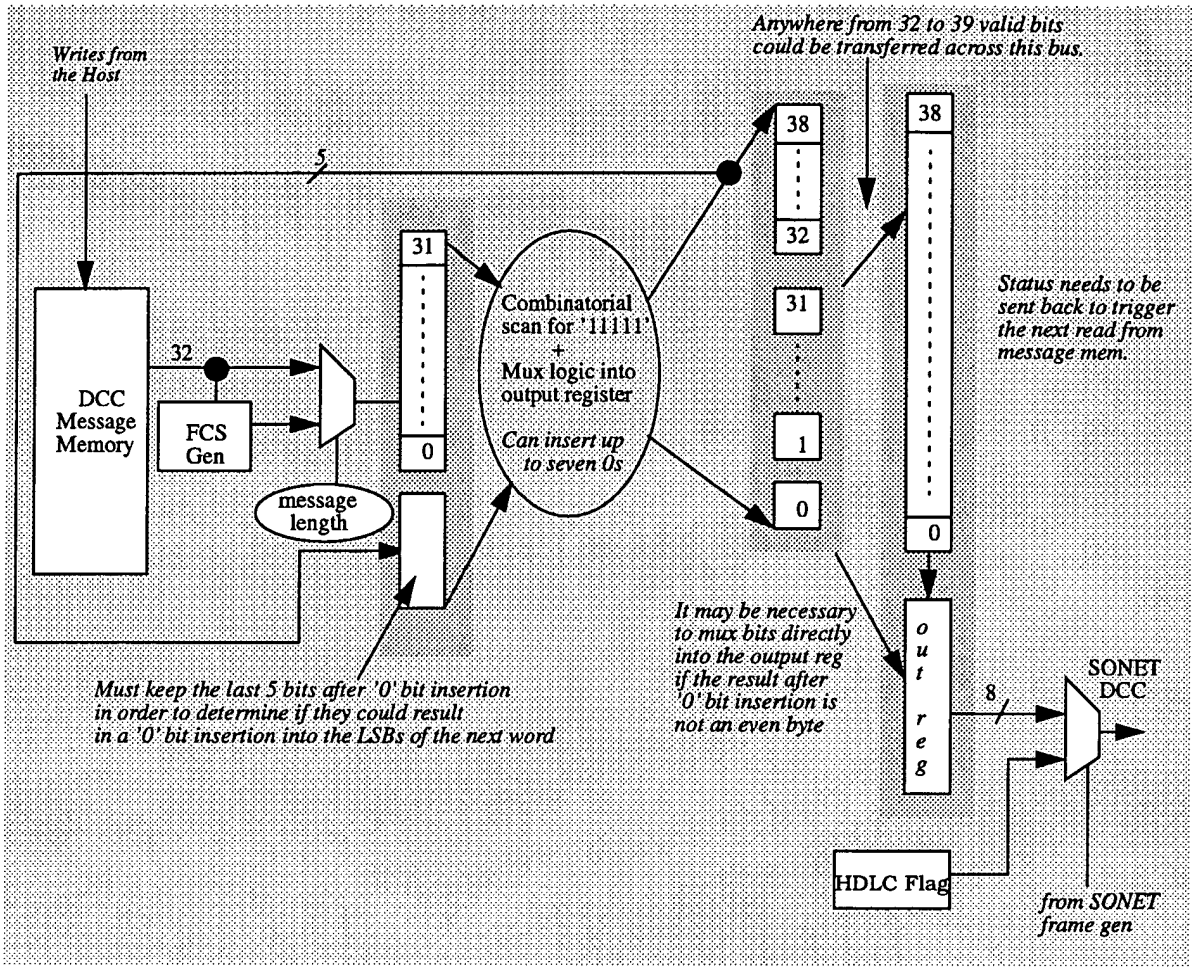


Figure 15-3: '0' bit insertion

The Flag sequence is transmitted on the DCC bytes when there is no message to send.

15.2 Rx Section & Line DCC

The SONET Data Communications Channels are accessed through the Host Processor Interface. The Rx SONET Overhead Processor Block is designed to receive and interpret 8 separate HDLC mapped data streams - one Section and one Line for each physical SONET port. A high level view of DCC support in the Service Processor is shown in Figure 15-4.

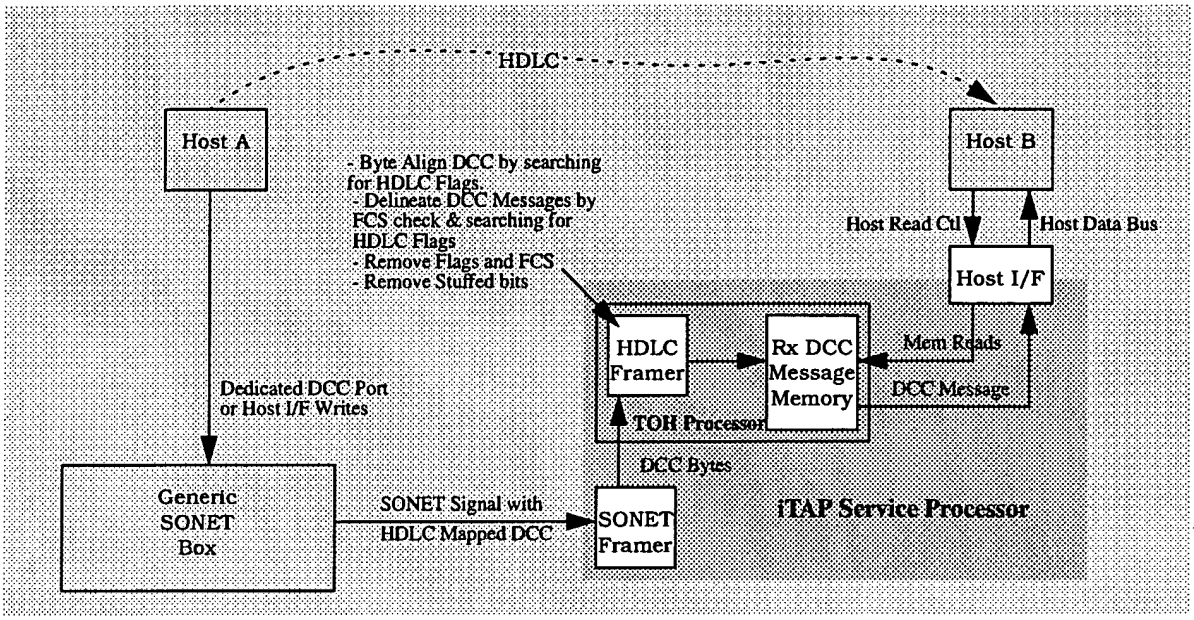


Figure 15-4: High Level View of Rx DCC

The Section DCC Bytes (D1, D2, D3) and the Line DCC bytes (D4-D12) of STS#1 of an STSN signal or of an STM signal are identified and routed to the HDLC Framer in the TOH Processor Block. There is a pair of HDLC framers for each SONET interface. One of the pair is used for Section DCC and the other for Line DCC.

The HDLC mapped data carried in the DCC bytes are **NOT** necessarily byte aligned. The Transmit SONET interface of the Service Processor does byte align DCC messages in the DCC byte locations, but it is not a requirement. The HDLC Framer searches for the Flag sequence in order to first byte align the DCC channel and then to delineate the messages carried in the DCC bytes. Figure 15-5 shows the HDLC framer logic. .

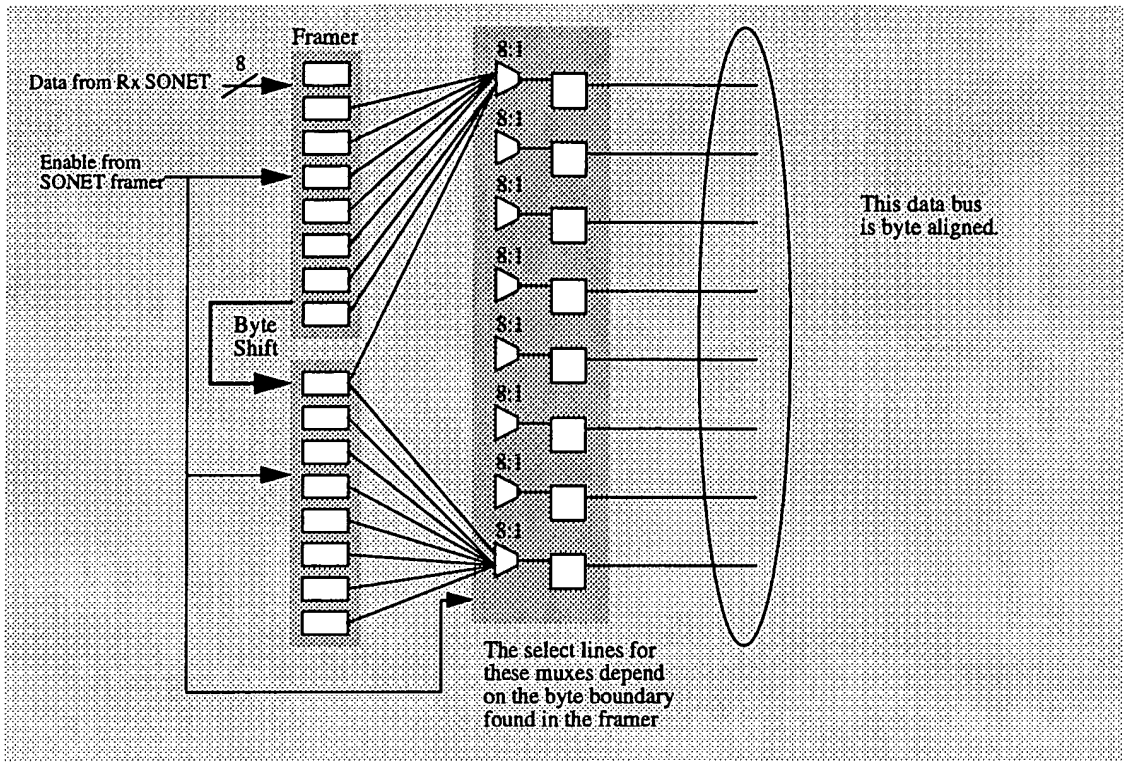


Figure 15-5: HDLC Framer

Once the framer identifies the byte boundary and finds a byte that is NOT Flag, the framer starts to extract the message embedded in the HDLC. The first NOT Flag byte following the last received Flag byte is the first byte of the DCC message. This first byte of the DCC message is the first byte used in the Frame Check Sequence (FCS) calculator. All of the bytes in the DCC message are used to calculate the FCS. The End of Message (EOM) is declared when the FCS is located in the data stream **AND** the next byte is a Flag byte. Both of these conditions must be detected before declaring EOM in order to avoid a false EOM declaration on the coincidental occurence of a correct FCS. The FCS can be located by comparing the resulting FCS value to the "good FCS" value indentified for the FCS algorithms.

The DCC message extracted from the HDLC is loaded into a RAM that is accessible by the Master Processor. When the end of the DCC message is detected a status bit will be set to indicate that a message is ready to be read and the message length in bytes will be available for reading by the processor.

1.0 Host Interface

This section is intended to serve as an interface specification that defines the message interface for the iTAP chipset. This interface defines a set of primitives that allows a host system to control and interact with both the Port Processor and Switch Element.

1.1 Description

The service interface is implemented through the use of mailboxes between an individual iTAP device and the host. Management of the queue and transfer of messages to and from the host are performed by the host interface. Buffer resources for the mailbox are contained in a single dual port memory. The physical description and organization of the mailboxes are described in a following section.

The request mailbox allows the host to post requests for operations as defined in the following interface specification. The standard message descriptor structure is an ?-byte message header followed by a variable length message . Much of the host activity through the request mailbox will be to configure the operational parameters.The format and definition of fields within each of the request messages are outlined in section ??.

During operation alerts/alarms are require notifying the host to activities and errors which are occurring during operation. Status mailboxes are used to post responses to host requests or alert the host to errors and events occurring during processing of the traffic. The status messages, like the request messages, are ?-byte message header followed by a variable length message . The format and definition of fields within each of the status messages are outlined in section ???.

The processing of the request and indicate message descriptors is performed completely by firmware which is running in the devices, and by the host driver. SInce the implementation of the service interface is completely in firmware/software, later changes may be made to optimize the interface or upgrade the commands to allow new features. This flexibility also allows customer specific requirements or value added features to be easily implemented.

2.0 Interface Description and Physical Organization

The host interface contains an integrated controller (tensilica processor) for transferring data and control information. The host interface is a message based interface between the host and the an iTAP devices, and is supported via mailboxes in the iTAP. A synchronous dual-port static ram is used to hold the mailbox entries. Each mailbox is ?? words in length. A number of operational and failure conditions can be reported to the host processor via messages. The host interface incorporates several FIFO buffers to allow masking of latency and significant improvement in throughput.

The communications procedure is as follows: the HOST or iTAP device writes a message to a mailbox. The write status field is toggled (HW/IW) to indicate to the recipient of the message and then the host interface asserts $\overline{\text{INTER}}$(R) (->iTAP device) or $\overline{\text{INTER}}$(L) (->HOST) is asserted. The reading entity reads the write status field, and then the message from the mailbox. The reading of the status field locations will deassert the INTER pin. After the complete message is read, the reader toggles the appropriate read status field, and then DPSRAM signals the writing entity via the $\overline{\text{INTER}}$ (L,R) pin. The original writer will then read the read status field to update the availability of the mailbox .

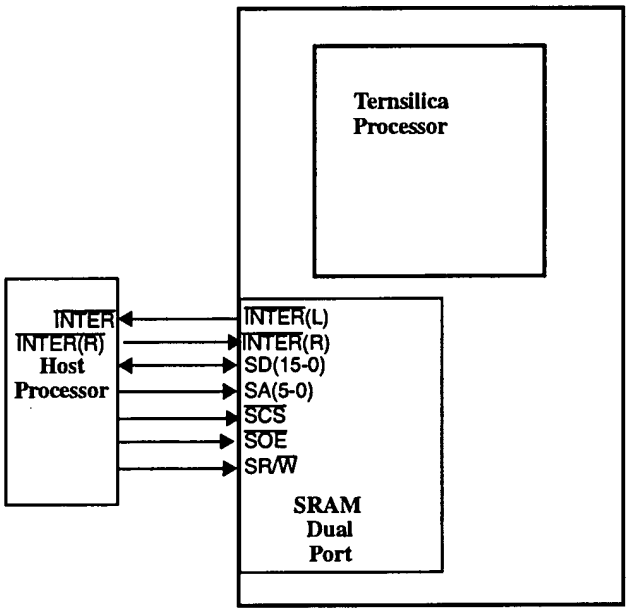


Figure 1. Host Processor Interface

2.1 Host Control Logic

The host interface control logic allows the external host processor to access on-chip control/status registers, and external control memory data structures. This allows the host software to configure, control and poll, and to communicate with firmware running on the internal tensilica cores. The control logic also implements read and write FIFO buffers that interface between the control logic and DMA controller and internal registers. The FIFO buffers allow burst writes and reads to be performed from the off-chip mailboxes to the local memory or on-chip registers.The host interface uses a ?-word write command/data FIFO, and a ?-word read command FIFO. These two FIFOs permit the host interface to implement a write-behind/fetch-ahead behavior: memory commands are buffered and memory read data words are prefetched to hide memory access latencies.Microprocessor interrupt pin INTER is asserted when the appropriate status field is written, and deasserted when the status field is read.

2.2 Mailbox Organization

There are 6 mailboxes which correspond to read/write to/from each individual internal processor to the host :

Addr								
	HOST -> iTAP (high priority)							
	iTAP -> HOST (high priority)							
	HOST -> iTAP (low priority)							
	iTAP -> HOST (low priority)							
	AW			AR				
	HW			HR				

Each Mailbox is a maximum of ? 32-bit words in length.

The mailbox static ram is a ?k memory organized as . Handshake status signals are in SRAM addresses NN

IW: 2-bits indicates current write status of iTAP to HOST mailboxes. Bits change on event basis i.e. are toggled. Organization is High, Low. iTAP updates this field.

IR: 2-bits indicates current read status of HOST to iTAP mailboxes. Bits change on event basis i.e. are toggled. Organization is High, Low. iTAP updates this field.

HW: 2-bits indicates current write status of iTAP to HOST mailboxes. Bits change on event basis i.e. are toggled. Organization is High, Low. Host updates this field.

HR: 2-bits indicates current read status of HOST to iTAP mailboxes. Bits change on event basis i.e. are toggled. Organization is High, Low. Host updates this field.